

Al Sweigart

# Routineaufgaben mit Python automatisieren

Praktische Programmierlösungen für Einsteiger



Papier  
plus<sup>+</sup>  
PDF.

Zu diesem Buch – sowie zu vielen weiteren dpunkt.büchern – können Sie auch das entsprechende E-Book im PDF-Format herunterladen. Werden Sie dazu einfach Mitglied bei dpunkt.plus<sup>+</sup>: [www.dpunkt.de/plus](http://www.dpunkt.de/plus)

**AI Sweigart**

# **Routineaufgaben mit Python automatisieren**

**Praktische Programmierlösungen für Einsteiger**



**dpunkt.verlag**

Al Sweigart

Lektorat: Dr. Michael Barabas

Fachgutachterin: Ari Lacenski

Copy-Editing: Petra Kienle

Übersetzung & Satz: G&U Language & Publishing Services GmbH, [www.gundu.com](http://www.gundu.com)

Herstellung: Nadine Thiele

Umschlaggestaltung: Helmut Kraus, [www.exclam.de](http://www.exclam.de)

nach der Originalvorlage von No Starch Press

Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, Paderborn

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:

Print 978-3-86490-353-3

PDF 978-3-86491-992-3

ePub 978-3-86491-993-0

mobi 978-3-86491-994-7

1. Auflage 2016

Copyright © 2016 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Copyright der amerikanischen Originalausgabe

© 2015 by Al Sweigart

Titel der Originalausgabe: Automate the Boring Stuff with Python

No Starch Press, Inc. · 38 Ringold Street, San Francisco, CA 94103 · <http://www.nostarch.com>

ISBN 978-159327-599-0

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

---

# Inhalt

<b>Der Autor .....</b>	<b>xxiv</b>
<b>Die Fachgutachterin .....</b>	<b>xxiv</b>
<b>Danksagung .....</b>	<b>xxv</b>
<b>Einleitung .....</b>	<b>1</b>
Für wen ist dieses Buch gedacht? .....	2
Programmierstil .....	3
Was ist Programmierung? .....	3
Was ist Python? .....	4
Programmierer müssen nicht viel Mathe können .....	4
Programmierung ist kreativ .....	5
Der Aufbau dieses Buchs .....	6
Python herunterladen und installieren .....	8
IDLE starten .....	9
Die interaktive Shell .....	9
Hilfe finden .....	10
Geschickte Fragen stellen .....	11
Zusammenfassung .....	12
<b>Teil 1     Grundlagen der Python-Programmierung</b>	<b>13</b>
<b>1 Grundlagen von Python .....</b>	<b>15</b>
Ausdrücke in die interaktive Shell eingeben .....	16
Die Datentypen für ganze Zahlen, Fließkommazahlen und Strings .....	19

Stringverkettung und -wiederholung .....	20
Werte in Variablen speichern .....	21
Zuweisungsanweisungen .....	21
Variablennamen .....	23
Ihr erstes Programm .....	24
Analyse des Programms .....	26
Kommentare .....	26
Die Funktion print() .....	26
Die Funktion input() .....	27
Den Benutzernamen ausgeben .....	27
Die Funktion len() .....	28
Die Funktionen str(), int() und float() .....	29
Zusammenfassung .....	32
Wiederholungsfragen .....	33
 <b>2 Flusssteuerung .....</b>	<b>35</b>
Boolesche Werte .....	36
Vergleichsoperatoren .....	37
Boolesche Operatoren .....	39
Binäre boolesche Operatoren .....	39
Der Operator not .....	40
Kombinierte Verwendung von booleschen und Vergleichsoperatoren .....	41
Elemente zur Flusssteuerung .....	42
Bedingungen .....	42
Codeblöcke .....	42
Programmausführung .....	43
Flusssteuerungsanweisungen .....	43
If-Anweisungen .....	43
Else-Anweisungen .....	44
Elif-Anweisungen .....	45
While-Schleifen .....	51
Break-Anweisungen .....	56
Continue-Anweisungen .....	57
For-Schleifen und die Funktion range() .....	61
Module importieren .....	65
From-import-Anweisungen .....	66
Programme mit sys.exit() vorzeitig beenden .....	66
Zusammenfassung .....	67
Wiederholungsfragen .....	67

<b>3 Funktionen .....</b>	<b>69</b>
Def-Anweisungen mit Parametern .....	71
Rückgabewerte und die Anweisung return .....	72
Der Wert None .....	73
Schlüsselwortargumente und print() .....	74
Lokaler und globaler Gültigkeitsbereich .....	75
Lokale Variablen können im globalen Gültigkeitsbereich nicht verwendet werden .....	76
Lokale Gültigkeitsbereiche können keine Variablen aus anderen lokalen Gültigkeitsbereichen verwenden .....	77
Globale Variablen können von einem lokalen Gültigkeitsbereich aus gelesen werden .....	78
Lokale und globale Variablen mit demselben Namen .....	78
Die Anweisung global .....	79
Ausnahmebehandlung .....	82
Ein kurzes Programm: Zahlen raten .....	84
Zusammenfassung .....	86
Wiederholungsfragen .....	87
Übungsprojekte .....	87
Die Collatz-Folge .....	87
Eingabevervalidierung .....	88
<b>4 Listen .....</b>	<b>89</b>
Der Datentyp für Listen .....	89
Einzelne Elemente aus einer Liste mithilfe des Index abrufen .....	90
Negative Indizes .....	92
Teillisten mit Slices abrufen .....	92
Die Länge einer Liste mit len() abrufen .....	93
Werte in einer Liste anhand des Index ändern .....	93
Listenverkettung und -wiederholung .....	94
Elemente mit del aus einer Liste entfernen .....	94
Listen verwenden .....	94
For-Loops für Listen .....	96
Die Operatoren in und not in .....	97
Der Trick mit der Mehrfachzuweisung .....	98
Erweiterte Zuweisungsoperatoren .....	99

Methoden . . . . .	100
Elemente in einer Liste mit der Methode index() finden . . . . .	100
Elemente mit den Methoden append() und insert() zu Listen hinzufügen . . . . .	101
Elemente mit remove() aus Listen entfernen . . . . .	102
Elemente in einer Liste mit sort() sortieren . . . . .	103
Beispielprogramm: Magic 8 Ball unter Verwendung einer Liste . . . . .	104
Listenähnliche Typen: Strings und Tupel . . . . .	106
Veränderbare und unveränderbare Datentypen . . . . .	106
Der Datentyp für Tupel . . . . .	109
Typen mit den Funktionen list() und tuple() umwandeln . . . . .	110
Verweise . . . . .	110
Verweise übergeben . . . . .	113
Die Funktionen copy() und deepcopy() des Moduls copy . . . . .	113
Zusammenfassung . . . . .	114
Wiederholungsfragen . . . . .	115
Übungsprojekte . . . . .	116
Kommacode . . . . .	116
Zeichenbildraster . . . . .	116
<b>5 Dictionarys und Datenstrukturen . . . . .</b>	<b>119</b>
Der Datentyp für Dictionarys . . . . .	119
Dictionarys und Listen im Vergleich . . . . .	120
Die Methoden keys(), values() und items() . . . . .	122
Das Vorhandensein eines Schlüssels oder Werts im Dictionary ermitteln . . . . .	123
Die Methode get() . . . . .	123
Die Methode setdefault() . . . . .	124
Saubere Ausgabe . . . . .	125
Datenstrukturen zur Modellierung realer Objekte . . . . .	127
Ein Tic-Tac-Toe-Brett . . . . .	128
Verschachtelte Dictionarys und Listen . . . . .	133
Zusammenfassung . . . . .	135
Wiederholungsfragen . . . . .	135
Übungsprojekte . . . . .	136
Inventar für ein Fantasyspiel . . . . .	136
Eine Funktion zum Hinzufügen von Listeninhalten zum Inventar-Dictionary . . . . .	136

<b>6 Stringbearbeitung .....</b>	<b>139</b>
Umgang mit Strings .....	139
Stringliterale .....	140
Strings indizieren und Slices entnehmen .....	143
Die Operatoren in und not in für Strings .....	144
Nützliche Stringmethoden .....	144
Die Stringmethoden upper(), lower(), isupper() und islower() .....	144
Die isX-Stringmethoden .....	146
Die Stringmethoden startswith() und endswith() .....	148
Die Methoden join() und split() .....	149
Text mit rjust(), ljust() und center() ausrichten .....	150
Weißraum mit strip(), rstrip() und lstrip() entfernen .....	152
Strings mit dem Modul pyperclip kopieren und einfügen .....	152
Projekt: Passwortsafe .....	153
Schritt 1: Programmdesign und Datenstrukturen .....	154
Schritt 2: Befehlszeilenargumente verarbeiten .....	154
Schritt 3: Das richtige Passwort kopieren .....	155
Projekt: Aufzählungspunkte zu einem Wiki-Markup hinzufügen .....	156
Schritt 1: Text von und zur Zwischenablage übertragen .....	157
Schritt 2: Textzeilen trennen und Sternchen hinzufügen .....	158
Schritt 3: Die veränderten Zeilen zusammenfügen .....	159
Zusammenfassung .....	159
Wiederholungsfragen .....	160
Übungsprojekt .....	161
Tabellenausgabe .....	161
<b>Teil 2     Aufgaben automatisieren</b>	<b>163</b>
<b>7 Mustervergleich mit regulären Ausdrücken .....</b>	<b>165</b>
Textmuster ohne reguläre Ausdrücke finden .....	166
Textmuster mithilfe regulärer Ausdrücke finden .....	168
Regex-Objekte erstellen .....	169
Vergleiche mit einem Regex-Objekt .....	170
Zusammenfassung: Mustervergleich mit regulären Ausdrücken .....	171

Weitere Möglichkeiten für den Mustervergleich mithilfe regulärer Ausdrücke .....	171
Gruppierung durch Klammern .....	171
Mithilfe der Pipe nach Übereinstimmungen mit mehreren Gruppen suchen .....	173
Optionale Übereinstimmung mit dem Fragezeichen .....	174
Mit dem Sternchen nach null oder mehr Übereinstimmungen suchen .....	174
Mit dem Pluszeichen nach einer oder mehr Übereinstimmungen suchen .....	175
Mit geschweiften Klammern nach einer genauen Zahl von Wiederholungen suchen .....	176
Gieriger und nicht gieriger Mustervergleich .....	177
Die Methode <code>findall()</code> .....	177
Zeichenklassen .....	178
Eigene Zeichenklassen bilden .....	179
Zirkumflex und Dollarzeichen .....	180
Das Jokerzeichen .....	181
Beliebige Übereinstimmungen mit Punkt-Stern finden .....	181
Zeilenumbrüche mit dem Punktsymbol finden .....	182
Übersicht über Regex-Symbole .....	183
Übereinstimmungen ohne Berücksichtigung der Groß- und Kleinschreibung .....	183
Strings mit der Methode <code>sub()</code> ersetzen .....	184
Umgang mit komplizierten regulären Ausdrücken .....	185
Die Variablen <code>re.IGNORECASE</code> , <code>re.DOTALL</code> und <code>re.VERBOSE</code> kombinieren .....	186
<b>Projekt:</b> Extraktionsprogramm für Telefonnummern und E-Mail-Adressen .....	186
Schritt 1: Einen regulären Ausdruck für Telefonnummern erstellen ..	187
Schritt 2: Einen regulären Ausdruck für E-Mail-Adressen erstellen ..	188
Schritt 3: Alle Überstimmungen im Inhalt der Zwischenablage finden ..	189
Schritt 4: Die gefundenen Übereinstimmungen zu einem String kombinieren .....	190
Das Programm ausführen .....	191
Ideen für ähnliche Programme .....	191
Zusammenfassung .....	192
Wiederholungsfragen .....	192

Übungsprojekte .....	194
Passwortstärke ermitteln .....	194
Regex-Version von strip() .....	194
<b>8 Dateien lesen und schreiben .....</b>	<b>195</b>
Dateien und Dateipfade .....	195
Backslash unter Windows und Schrägstrich unter OS X und Linux ..	196
Das aktuelle Arbeitsverzeichnis .....	197
Absolute und relative Pfade .....	198
Neue Ordner mit os.makedirs() erstellen .....	199
Das Modul os.path .....	200
Absolute und relative Pfade verwenden .....	200
Dateigrößen und Ordnerinhalte ermitteln .....	202
Die Gültigkeit von Pfaden prüfen .....	203
Dateien lesen und schreiben .....	204
Dateien mit der Funktion open() öffnen .....	205
Die Inhalte einer Datei lesen .....	206
Dateien schreiben .....	207
Variablen mit dem Modul shelve speichern .....	208
Variablen mit der Funktion pprint.pformat() speichern .....	210
<b>Projekt: Zufallsgenerator für Tests .....</b>	<b>211</b>
Schritt 1: Die Daten für den Test in einem Dictionary speichern .....	211
Schritt 2: Die Fragebogendatei erstellen und die Fragen mischen ..	213
Schritt 3: Die Auswahl der möglichen Antworten zusammenstellen ..	214
Schritt 4: Den Inhalt der Dateien für die Frage- und Lösungsbogen schreiben .....	215
<b>Projekt: Mehrfach-Zwischenablage .....</b>	<b>216</b>
Schritt 1: Kommentare und Vorbereitungen für die Shelf-Daten .....	217
Schritt 2: Den Inhalt der Zwischenablage unter einem Schlüsselwort speichern .....	218
Schritt 3: Schlüsselwörter auflisten und Inhalte laden .....	219
Zusammenfassung .....	220
Wiederholungsfragen .....	220
Übungsprojekte .....	221
Erweiterte Mehrfach-Zwischenablage .....	221
Lückentextspiel .....	221
Regex-Suche .....	222

<b>9 Dateien verwalten .....</b>	<b>223</b>
Das Modul shutil .....	224
Dateien und Ordner kopieren .....	224
Dateien und Ordner verschieben und umbenennen .....	225
Dateien und Ordner unwiederbringlich löschen .....	227
Sicheres Löschen mit dem Modul send2trash .....	228
Einen Verzeichnisbaum durchlaufen .....	228
Dateien mit der Methode zipfile komprimieren .....	230
ZIP-Dateien lesen .....	231
ZIP-Dateien entpacken .....	232
ZIP-Dateien erstellen und Inhalte hinzufügen .....	233
<b>Projekt: Amerikanische Datumsangaben in europäische ändern .....</b>	<b>233</b>
Schritt 1: Einen regulären Ausdruck für amerikanische Datumsangaben definieren .....	234
Schritt 2: Die einzelnen Teile der Datumsangabe in den Dateinamen ermitteln .....	235
Schritt 3: Die neuen Dateinamen zusammenstellen und die Dateien umbenennen .....	237
Vorschläge für ähnliche Programme .....	238
<b>Projekt: Einen Ordner in einer ZIP-Datei sichern .....</b>	<b>238</b>
Schritt 1: Die Namen der ZIP-Dateien bestimmen .....	238
Schritt 2: Die neue ZIP-Datei erstellen .....	240
Schritt 3: Den Verzeichnisbaum durchlaufen und Inhalte zur ZIP-Datei hinzufügen .....	240
Vorschläge für ähnliche Programme .....	242
Zusammenfassung .....	242
Wiederholungsfragen .....	243
Übungsprojekte .....	243
Selektives Kopieren .....	243
Nicht mehr benötigte Dateien löschen .....	243
Lücken entfernen .....	244
<b>10 Debugging .....</b>	<b>245</b>
Ausnahmen auslösen .....	246
Traceback als String abrufen .....	248
Zusicherungen (Assertions) .....	249
Zusicherungen in einem Ampelsimulator .....	250
Zusicherungen deaktivieren .....	252

Protokollierung .....	252
Das Modul logging verwenden .....	252
Kein Debugging mit print() .....	254
Protokolliergrade .....	255
Die Protokollierung deaktivieren .....	256
Protokollierung in eine Datei .....	257
Der Debugger von IDLE .....	257
Go .....	258
Step .....	258
Over .....	258
Out .....	259
Quit .....	259
Debugging eines Additionsprogramms .....	259
Haltepunkte .....	262
Zusammenfassung .....	263
Wiederholungsfragen .....	264
Übungsprojekt .....	264
Münzwurfprogramm .....	264
<b>11 Web Scraping .....</b>	<b>267</b>
Projekt: mapIt.py mit dem Modul webbrowser .....	268
Schritt 1: Den URL herausfinden .....	269
Schritt 2: Befehlszeilenargumente verarbeiten .....	269
Schritt 3: Den Inhalt der Zwischenablage verarbeiten und den Browser starten .....	270
Vorschläge für ähnliche Programme .....	271
Dateien mithilfe des Moduls requests aus dem Web herunterladen .....	271
Eine Webseite mit der Funktion requests.get() herunterladen .....	272
Nach Fehlern suchen .....	273
Heruntergeladene Dateien auf der Festplatte speichern .....	274
HTML .....	275
Quellen zu HTML .....	276
Ein kleiner Auffrischungskurs .....	276
Den HTML-Quellcode einer Webseite einsehen .....	277
Die Entwickertools des Browsers öffnen .....	278
HTML-Elemente mithilfe der Entwickertools finden .....	280

HTML mit dem Modul BeautifulSoup durchsuchen . . . . .	281
Ein BeautifulSoup-Objekt aus dem HTML-Text erstellen . . . . .	281
Elemente mit der Methode select() finden . . . . .	282
Daten aus den Attributen eines Elements abrufen . . . . .	284
<b>Projekt: Google-Suche »Auf gut Glück« . . . . .</b>	<b>285</b>
Schritt 1: Die Befehlszeilenargumente abrufen und die Suchergebnisseite anfordern . . . . .	285
Schritt 2: Alle Ergebnisse finden . . . . .	286
Schritt 3: Browserstabs für jedes Suchergebnis öffnen . . . . .	287
Vorschläge für ähnliche Programme . . . . .	288
<b>Projekt: Alle XKCD-Comics herunterladen . . . . .</b>	<b>288</b>
Schritt 1: Den Aufbau des Programms festlegen . . . . .	290
Schritt 2: Die Webseite herunterladen . . . . .	291
Schritt 3: Das Bild des Comics finden und herunterladen . . . . .	291
Schritt 4: Das Bild speichern und den vorherigen Comic suchen . . . . .	292
Vorschläge für ähnliche Programme . . . . .	294
<b>Den Browser mit dem Modul Selenium steuern . . . . .</b>	<b>294</b>
Einen seleniumgesteuerten Browser starten . . . . .	295
Elemente auf der Seite finden . . . . .	295
Auf Links klicken . . . . .	297
Formulare ausfüllen und absenden . . . . .	298
Die Betätigung von Sondertasten simulieren . . . . .	298
Auf Browserschaltflächen klicken . . . . .	299
Weitere Informationen über Selenium . . . . .	300
<b>Zusammenfassung . . . . .</b>	<b>300</b>
<b>Wiederholungsfragen . . . . .</b>	<b>300</b>
<b>Übungsprojekte . . . . .</b>	<b>301</b>
E-Mail-Programm für die Befehlszeile . . . . .	301
Download-Programm für Fotowebsites . . . . .	301
2048 . . . . .	302
Linküberprüfung . . . . .	302
<b>12 Arbeiten mit Excel-Arbeitsblättern . . . . .</b>	<b>303</b>
<b>Excel-Dokumente . . . . .</b>	<b>304</b>
<b>Das Modul openpyxl installieren . . . . .</b>	<b>304</b>
<b>Excel-Dokumente lesen . . . . .</b>	<b>304</b>
Excel-Dokumente mit OpenPyXL öffnen . . . . .	305
Arbeitsblätter aus der Arbeitsmappe abrufen . . . . .	306

Zellen in Arbeitsblättern ansprechen .....	306
Umrechnen zwischen Kennbuchstaben und Nummern .....	308
Zeilen und Spalten eines Arbeitsblatts abrufen .....	309
Arbeitsmappen, Arbeitsblätter und Zellen .....	311
<b>Projekt:</b> Daten in einer Arbeitsmappe lesen .....	311
Schritt 1: Die Daten der Arbeitsmappe lesen .....	312
Schritt 2: Die Datenstruktur füllen .....	313
Schritt 3: Die Ergebnisse in eine Datei schreiben .....	315
Vorschläge für ähnliche Programme .....	316
Excel-Dokumente schreiben .....	317
Excel-Dokumente erstellen und speichern .....	317
Arbeitsblätter erstellen und entfernen .....	318
Werte in Zellen schreiben .....	319
<b>Projekt:</b> Ein Arbeitsblatt aktualisieren .....	319
Schritt 1: Eine Datenstruktur mit den neuen Informationen einrichten .....	320
Schritt 2: Alle Zeilen prüfen und die falschen Preise korrigieren .....	321
Vorschläge für ähnliche Programme .....	322
Die Schrift in den Zellen gestalten .....	323
Font-Objekte .....	324
Formeln .....	325
Das Erscheinungsbild von Zeilen und Spalten festlegen .....	327
Zeilenhöhe und Spaltenbreite festlegen .....	327
Zellen verbinden und aufteilen .....	328
Bereiche fixieren .....	329
Diagramme .....	330
Zusammenfassung .....	332
Wiederholungsfragen .....	333
Übungsprojekte .....	334
Multiplikationstabellen erstellen .....	334
Leere Zeilen einfügen .....	334
Zellen transponieren .....	335
Textdateien in Arbeitsblätter umwandeln .....	336
Arbeitsblätter in Textdateien umwandeln .....	336

<b>13 Arbeiten mit PDF- und Word-Dokumenten .....</b>	<b>337</b>
PDF-Dokumente .....	337
Text aus PDFs entnehmen .....	338
PDFs entschlüsseln .....	340
PDFs erstellen .....	340
Projekt: Ausgewählte Seiten aus mehreren PDFs kombinieren .....	346
Schritt 1: Alle PDF-Dateien finden .....	346
Schritt 2: Die einzelnen PDFs öffnen .....	347
Schritt 3: Die einzelnen Seiten hinzufügen .....	348
Schritt 4: Die Ergebnisse speichern .....	349
Vorschläge für ähnliche Programme .....	349
Word-Dokumente .....	350
Word-Dokumente lesen .....	351
Den kompletten Text einer .docx-Datei abrufen .....	352
Absätze und Run-Objekte formatieren .....	353
Word-Dokumente mit anderen als den Standardformaten erstellen ..	355
Run-Attribute .....	356
Word-Dokumente schreiben .....	358
Überschriften hinzufügen .....	360
Zeilenwechsel und Seitenumbrüche hinzufügen .....	360
Bilder einfügen .....	361
Zusammenfassung .....	362
Wiederholungsfragen .....	362
Übungsprojekte .....	363
PDF-Paranoia .....	363
Personalisierte Einladungen als Word-Dokument .....	363
Brute-Force-Passwortknacker für PDFs .....	364
<b>14 Arbeiten mit CSV-Dateien und JSON-Daten .....</b>	<b>367</b>
Das Modul csv .....	368
Reader-Objekte .....	369
Daten in einer for-Schleife aus Reader-Objekten lesen .....	370
Writer-Objekte .....	371
Die Schlüsselwortargumente delimiter und lineterminator .....	372
Projekt: Kopfzeilen aus CSV-Dateien entfernen .....	373
Schritt 1: Alle CSV-Dateien durchlaufen .....	373
Schritt 2: Die CSV-Datei lesen .....	374
Schritt 3: Die CSV-Datei ohne die erste Zeile schreiben .....	375
Vorschläge für ähnliche Programme .....	376

JSON und APIs .....	376
Das Modul json .....	378
JSON-Daten mit der Funktion loads() laden .....	378
JSON-Daten mit der Funktion dumps() schreiben .....	378
<b>Projekt:</b> Die aktuellen Wetterdaten abrufen .....	379
Schritt 1: Den Standort aus dem Befehlszeilenargument entnehmen ..	379
Schritt 2: Die JSON-Daten herunterladen .....	380
Schritt 3: JSON-Daten laden und die Wettervorhersage ausgeben ..	381
Vorschläge für ähnliche Programme .....	382
Zusammenfassung .....	383
Wiederholungsfragen .....	383
Übungsprojekt .....	384
Excel-in-CSV-Konverter .....	384
<b>15 Zeit einhalten, Aufgaben zeitlich planen und Programme starten .....</b>	<b>385</b>
Das Modul time .....	386
Die Funktion time.time() .....	386
Die Funktion time.sleep() .....	387
Zahlen runden .....	388
<b>Projekt:</b> Superstoppuhr .....	389
Schritt 1: Das Programm auf die Zeitmessung vorbereiten .....	390
Schritt 2: Intervalldauern messen und anzeigen .....	390
Vorschläge für ähnliche Programme .....	391
Das Modul datetime .....	392
Der Datentyp timedelta .....	394
Anhalten bis zu einem bestimmten Zeitpunkt .....	395
datetime-Objekte in Strings umwandeln .....	396
Strings in datetime-Objekte umwandeln .....	397
Die Zeitfunktionen von Python im Überblick .....	398
Multithreading .....	399
Argumente an die Zielfunktion eines Threads übergeben .....	401
Probleme der Nebenläufigkeit .....	402
<b>Projekt:</b> Multithread-Version des XKCD-Download-Programms .....	402
Schritt 1: Eine Funktion für den Download verwenden .....	403
Schritt 2: Threads erstellen und starten .....	404
Schritt 3: Auf das Ende aller Threads warten .....	405

Andere Programme von Python aus starten . . . . .	405
Befehlszeilenargumente an Popen() übergeben . . . . .	407
Taskplaner, launchd und cron . . . . .	408
Websites mit Python aufrufen . . . . .	408
Andere Python-Skripte ausführen . . . . .	408
Dateien in ihren Standardanwendungen öffnen . . . . .	409
Projekt: Ein einfaches Countdown-Programm . . . . .	411
Schritt 1: Der Countdown . . . . .	411
Schritt 2: Die Klangdatei abspielen . . . . .	412
Vorschläge für ähnliche Programme . . . . .	413
Zusammenfassung . . . . .	413
Wiederholungsfragen . . . . .	414
Übungspunkte . . . . .	414
Elegantere Stoppuhr . . . . .	414
Webcomic-Download-Programm mit Zeitplanung . . . . .	415
<b>16 E-Mails und Textnachrichten senden . . . . .</b>	<b>417</b>
SMTP . . . . .	418
E-Mails senden . . . . .	418
Verbindung mit einem SMTP-Server aufnehmen . . . . .	419
Die »Hallo«-Nachricht an den SMTP-Server senden . . . . .	420
Die TLS-Verschlüsselung einleiten . . . . .	421
Am SMTP-Server anmelden . . . . .	421
Eine E-Mail senden . . . . .	422
Die Verbindung zum SMTP-Server trennen . . . . .	422
IMAP . . . . .	423
E-Mails mit IMAP abrufen und löschen . . . . .	423
Verbindung mit einem IMAP-Server aufnehmen . . . . .	424
Am IMAP-Server anmelden . . . . .	425
Nach E-Mails suchen . . . . .	425
E-Mails abrufen und als gelesen markieren . . . . .	430
E-Mail-Adressen aus einer Rohnachricht gewinnen . . . . .	431
Den Rumpf aus einer Rohnachricht gewinnen . . . . .	432
E-Mails löschen . . . . .	433
Die Verbindung zum IMAP-Server trennen . . . . .	434

<b>Projekt:</b> E-Mails über ausstehende Mitgliedsbeiträge senden .....	435
Schritt 1: Die Excel-Datei öffnen .....	435
Schritt 2: Alle säumigen Mitglieder finden .....	437
Schritt 3: Personalisierte E-Mail-Mahnungen senden .....	437
Textnachrichten mit Twilio senden .....	439
Ein Twilio-Konto einrichten .....	440
Textnachrichten senden .....	440
<b>Projekt:</b> Das Modul »Just Text Me« .....	443
Zusammenfassung .....	444
Wiederholungsfragen .....	444
Übungsprojekte .....	445
Zufällige Zuweisung von Arbeiten .....	445
Regenschirmhinweis .....	445
Automatischer Entregistrierer .....	446
Den Computer per E-Mail steuern .....	446
<b>17 Bildbearbeitung</b> .....	<b>449</b>
Grundlagen zur Bilddarstellung auf Computern .....	450
Farben und RGBA-Werte .....	450
Koordinaten und Rechtektupel .....	452
Bildbearbeitung mit Pillow .....	453
Arbeiten mit dem Datentyp Image .....	455
Bilder beschneiden .....	456
Bilder kopieren und in andere Bilder einfügen .....	457
Die Bildgröße ändern .....	460
Bilder drehen und spiegeln .....	461
Einzelne Pixel ändern .....	463
<b>Projekt:</b> Ein Logo hinzufügen .....	464
Schritt 1: Das Logobild öffnen .....	465
Schritt 2: Alle Dateien durchlaufen und die Bilder öffnen .....	466
Schritt 3: Die Bildgröße ändern .....	467
Schritt 4: Logo hinzufügen und Änderungen speichern .....	469
Vorschläge für ähnliche Programme .....	470

Bilder zeichnen .....	471
Formen zeichnen .....	471
Text zeichnen .....	473
Zusammenfassung .....	475
Wiederholungsfragen .....	476
Übungsprojekte .....	476
Das Logoprogramm erweitern und verbessern .....	476
Fotoordner auf der Festplatte finden .....	477
Personalisierte Tischkarten .....	478
<b>18 Tastatur und Maus mit GUI-Automatisierung steuern .....</b>	<b>481</b>
Das Modul PyAutoGUI installieren .....	482
Kleine Probleme beheben .....	482
Beenden durch Abmelden .....	483
Pausen und Sicherungen .....	483
Den Mauszeiger steuern .....	484
Den Mauszeiger bewegen .....	485
Die Position des Mauszeigers abrufen .....	486
<b>Projekt: Wo ist mein Mauszeiger? .....</b>	<b>486</b>
Schritt 1: Das Modul importieren .....	487
Schritt 2: Den Beendigungscode und die Endlosschleife einrichten ..	487
Schritt 3: Die Koordinaten des Mauszeigers abrufen und anzeigen ..	488
Mausinteraktionen .....	489
Klicken .....	489
Ziehen .....	490
Scrollen .....	492
Auf dem Bildschirm arbeiten .....	493
Einen Screenshot aufnehmen .....	493
Einen Screenshot analysieren .....	494
<b>Projekt: Das Programm mouseNow erweitern .....</b>	<b>495</b>
Bilderkennung .....	495
Die Tastatur steuern .....	497
Strings von der Tastatur senden .....	497
Tastennamen .....	498
Tasten drücken und loslassen .....	500
Tastenkombinationen .....	500

Übersicht über die Funktionen von PyAutoGUI .....	501
<b>Projekt: Formulare automatisch ausfüllen .....</b>	<b>502</b>
Schritt 1: Den Ablauf herausfinden .....	503
Schritt 2: Die Koordinaten ermitteln .....	504
Schritt 3: Daten eingeben .....	506
Schritt 4: Auswahllisten und Optionsschalter .....	508
Schritt 5: Das Formular absenden und warten .....	509
Zusammenfassung .....	510
Wiederholungsfragen .....	510
Übungsprojekte .....	511
Beschäftigung vortäuschen .....	511
Instant-Messenger-Bot .....	511
Tutorial für einen Spiele-Bot .....	512
<b>Anhang</b>	<b>513</b>
<b>A Drittanbietermodule installieren .....</b>	<b>513</b>
Pip .....	513
Drittanbietermodule installieren .....	514
<b>B Programme ausführen .....</b>	<b>517</b>
Die Shebang-Zeile .....	517
Python-Programme unter Windows ausführen .....	518
Python-Programme unter OS X und Linux ausführen .....	519
Python-Programme mit ausgeschalteten Zusicherungen ausführen .....	519
<b>C Antworten auf die Wiederholungsfragen .....</b>	<b>521</b>
Kapitel 1 .....	521
Kapitel 2 .....	522
Kapitel 3 .....	524
Kapitel 4 .....	524
Kapitel 5 .....	525
Kapitel 6 .....	526
Kapitel 7 .....	526
Kapitel 8 .....	527

---

Kapitel 9 .....	528
Kapitel 10 .....	528
Kapitel 11 .....	529
Kapitel 12 .....	530
Kapitel 13 .....	531
Kapitel 14 .....	532
Kapitel 15 .....	532
Kapitel 16 .....	533
Kapitel 17 .....	533
Kapitel 18 .....	534
<b>Stichwortverzeichnis .....</b>	<b>535</b>

*Für meinen Neffen Jack*

## **Der Autor**

Der Softwareentwickler und Fachbuchautor Al Sweigart lebt in San Francisco. Python ist seine Lieblingsprogrammiersprache und er hat bereits mehrere Open-Source-Module dafür entwickelt. Seine anderen Bücher sind auf seiner Website <http://www.inventwithpython.com/> unter einer Creative-Commons-Lizenz kostenlos erhältlich. Seine Katze wiegt 14 Pfund.

## **Die Fachgutachterin**

Ari Lacenski ist Entwicklerin für Android-Anwendungen und Python-Software. Sie wohnt in San Francisco, wo sie auf <http://gradlewhy.ghost.io/> über Android-Programmierung schreibt. Sie ist Mentorin bei Women Who Code und Folkgitarristin.

## **Danksagung**

Um dieses Buch zu schreiben, benötigte ich die Hilfe vieler Menschen. Ich möchte Bill Pollock, meinen Lektoren Laurel Chun, Leslie Shen, Greg Poulos und Jennifer Griffith-Delgado und allen anderen Mitarbeitern bei No Starch Press für ihre unerschöpfliche Hilfe danken. Danke auch an meine Fachgutachterin Ari Lacenski für hervorragende Vorschläge und Verbesserungen und die Unterstützung.

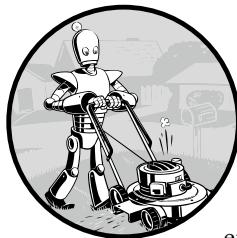
Vielen Dank auch meinem wohlmeinenden Diktator auf Lebenszeit, Guido van Rossum, und allen anderen bei der Python Software Foundation für ihre großartige Arbeit. Die Python-Community ist die beste, die ich in der technischen Branche gefunden habe.

Schließlich möchte ich noch meiner Familie, meinen Freunden und der Clique bei Shotwell's dafür danken, weil sie es mir nicht übelgenommen haben, dass ich beim Schreiben dieses Buchs dauernd beschäftigt war. Dankeschön!



---

## Einleitung



»Du hast gerade in zwei Stunden das erledigt, woran wir drei sonst zwei Tage lang sitzen!«

Mein Mitbewohner in den frühen 2000er-Jahren arbeitete bei einem Elektronikhändler, bei dem gelegentlich ein Arbeitsblatt mit

Tausenden von Produktpreisen eines Konkurrenten auftauchte. Drei Mitarbeiter druckten dieses Arbeitsblatt dann auf einem dicken Stapel Papier aus und teilten diesen unter sich auf. Für jeden Produktpreis schlugen sie den Preis ihres eigenen Arbeitgebers nach und notierten alle Produkte, die die Konkurrenz billiger anbot. Damit waren sie gewöhnlich zwei Tage beschäftigt.

»Wenn ich die Originaldatei bekomme, kann ich ein Programm schreiben, das die Arbeit für euch erledigt«, schlug mein Mitbewohner ihnen vor, als er sah, wie sie inmitten eines Riesenhaufens Papier auf dem Fußboden hockten.

Nach ein paar Stunden hatte er ein kurzes Programm geschrieben, das die Preisliste des Konkurrenten aus der Datei auslas, die Produkte in der Datenbank des Elektronikladens nachschlug und einen Vermerk machte, wenn die Konkurrenz billiger war. Er war immer noch ein Anfänger in Sachen Programmierung

und den Großteil dieser Stunden hatte er damit zugebracht, die Dokumentation in einem Programmierbuch nachzuschlagen. Die Ausführung des fertigen Programms dauerte nur wenige Sekunden. An dem Tag gönnten sich mein Mitbewohner und seine Kollegen eine besonders lange Mittagspause.

Das zeigt das Potenzial der Programmierung. Ein Computer ist wie ein Schweizer Messer und lässt sich für zahllose Aufgaben einrichten. Viele Leute bringen Stunden mit Klicken und Tippen zu, um monotone Aufgaben durchzuführen, ohne zu ahnen, dass der Computer die Arbeit in wenigen Sekunden erledigen könnte, wenn er nur die richtigen Anweisungen dafür bekäme.

## Für wen ist dieses Buch gedacht?

Viele Hilfsmittel, die wir heutzutage verwenden, stützen sich auf Software: Fast jeder nutzt Social Networks zur Kommunikation, die Telefone vieler Menschen enthalten Computer mit Internetzugriff und für die meisten Büroarbeiten ist Computerarbeit erforderlich. Daher ist die Nachfrage nach Personen, die programmieren können, sprunghaft angestiegen. Unzählige Bücher, interaktive Webtutorials und Trainingslager für Entwickler werden mit dem Versprechen beworben, ehrgeizige Anfänger zu Softwareingenieuren zu machen, die sechsstellige Gehälter verlangen können.

Dieses Buch ist nicht für diese Leute, sondern für alle anderen.

Mit diesem Buch allein können Sie nicht zu einem professionellen Softwareentwickler werden, genauso wenig wie ein paar Gitarrenstunden Sie zu einem Rockstar machen. Wenn Sie aber Büroangestellter, Administrator oder Akademiker oder irgendjemand anderes sind, der zur Arbeit oder zum Vergnügen einen Computer verwendet, so werden Sie hier die Grundlagen der Programmierung kennenlernen, um einfache Aufgaben wie die folgenden zu automatisieren:

- Tausende von Dateien verschieben und umbenennen und in Ordner sortieren,
- Onlineformulare ausfüllen, ohne Text eingeben zu müssen,
- Dateien von einer Website herunterladen oder Texte von dort kopieren, sobald dort neues Material bereitgestellt wird,
- sich von Ihrem Computer per SMS benachrichtigen lassen,
- Excel-Arbeitsblätter bearbeiten und formatieren,
- nach neuen E-Mails suchen und vorformulierte Antworten senden.

Diese Aufgaben sind einfach, aber zeitraubend, und sie sind häufig so trivial oder so spezifisch, dass es keine fertige Software dafür gibt. Mit einigen Programmierkenntnissen können Sie Ihren Computer diese Aufgaben für sich erledigen lassen.

## Programmierstil

Dieses Buch ist nicht als Nachschlagewerk gedacht, sondern als Anleitung für Anfänger. Der Programmierstil verstößt manchmal gegen die üblichen Richtlinien (beispielsweise werden in einigen Programmen globale Variablen verwendet), aber das ist ein Kompromiss, um das Lernen zu vereinfachen. Dieses Buch ist dazu gedacht, Wegwerfcode für einzelne Aufgaben zu schreiben, weshalb wir nicht viel Mühe auf Stil und Eleganz verwenden. Auch anspruchsvolle Programmierkonzepte wie Objektorientierung, Listenabstraktion und Generatoren werden hier aufgrund ihrer Kompliziertheit nicht behandelt. Altgediente Programmierer werden den Code sicherlich ändern wollen, um die Effizienz zu erhöhen, aber in diesem Buch geht es darum, Programme mit so wenig Aufwand wie möglich zum Laufen zu bekommen.

## Was ist Programmierung?

In Filmen und Fernsehserien werden Programmierer oft als Leute dargestellt, die wie rasend auf einer Tastatur herumtippen, um kryptische Folgen von Nullen und Einsen auf leuchtenden Bildschirmen erscheinen zu lassen. In Wirklichkeit ist moderne Programmierung aber nicht so geheimnisvoll. *Programmierung* ist einfach die Eingabe von Anweisungen, die der Computer ausführen soll. Diese Anweisungen können dazu dienen, mit Zahlen zu rechnen, Text zu ändern, Informationen in Dateien nachzuschlagen oder über das Internet mit anderen Computern zu kommunizieren.

Alle Programme bestehen aus einfachen Anweisungen, die die Grundbausteine darstellen. Einige der gebräuchlichsten dieser Anweisungen besagen, auf Deutsch übersetzt, Folgendes:

»Mach dies; dann mach das.«

»Wenn diese Bedingung wahr ist, dann führe diese Aktion aus; anderenfalls jene Aktion.«

»Mach dies soundso oft.«

»Mach dies, solange die Bedingung wahr ist.«

Diese Bausteine können Sie kombinieren, um auch kompliziertere Entscheidungen zu treffen. Im folgenden Beispiel sehen Sie die Programmieranweisungen – den *Quellcode* – für ein einfaches Programm in der Programmiersprache Python. Die Software Python führt die einzelnen Codezeilen vom Anfang bis zum Ende aus. (Manche Zeilen werden nur ausgeführt, wenn (*if*) eine Bedingung wahr ist (*true*); anderenfalls (*else*) führt Python eine andere Zeile aus.)

```

passwordFile = open('SecretPasswordFile.txt')    ❶
secretPassword = passwordFile.read()    ❷
print('Enter your password.')    ❸
typedPassword = input()
if typedPassword == secretPassword:    ❹
    print('Access granted')    ❺
    if typedPassword == '12345':    ❻
        print('That password is one that an idiot puts on their luggage.')    ❼
else:
    print('Access denied')    ❽

```

Auch wenn Sie noch nicht viel von Programmierung verstehen, können Sie vielleicht doch erraten, was der vorstehende Code bewirkt. Als Erstes wird die Datei *SecretPasswordFile.txt* geöffnet (❶) und das geheime Passwort gelesen (❷). Danach wird der Benutzer aufgefordert, ein Passwort einzugeben (über die Tastatur) (❸). Die beiden Passwörter werden verglichen (❹) und wenn sie identisch sind, gibt das Programm auf dem Bildschirm die Meldung *Access granted* (»Zugang gewährt«) aus (❺). Danach prüft das Programm, ob das Passwort *12345* lautet (❻); wenn ja, gibt es dem Benutzer den dezenten Hinweis, dass dies nicht gerade die ideale Wahl für ein Passwort ist (❼). Sind die Passwörter nicht identisch, gibt das Programm *Access denied* (»Zugriff verweigert«) aus (❽).

## Was ist Python?

Der Begriff *Python* bezeichnet die Programmiersprache Python (deren Syntaxregeln festlegen, was als gültiger Python-Code angesehen wird) und den Python-Interpreter, eine Software, die den (in der Sprache Python geschriebenen) Code liest und dessen Anweisungen ausführt. Den Python-Interpreter können Sie kostenlos von <http://python.org/> herunterladen, wobei es Versionen für Linux, OS X und Windows gibt.

Der Name Python ist übrigens nicht von der Schlange abgeleitet, sondern von der surrealistischen britischen Komikergruppe Monty Python. Python-Programmierer werden liebevoll »Pythonistas« genannt und Tutorials sowie Dokumentation zu Python stecken voller Anspielungen sowohl auf Monty Python als auch auf Schlangen.

## Programmierer müssen nicht viel Mathe können

Wenn mir jemand erklärt, warum er Angst davor hat, Programmieren zu lernen, geht es meistens darum, dass er glaubt, dazu müsste man sehr gut in Mathematik sein. In Wirklichkeit ist zur Programmierung meistens nicht mehr Mathe als einfache Grundrechenarten erforderlich. Programmieren lässt sich in diesem Punkt

sogar mit dem Lösen von Sudoku-Rätseln vergleichen. Dazu müssen Sie in jede Zeile, jede Spalte und jedes innere 3x3-Quadrat des gesamten 9x9-Feldes die Zahlen von 1 bis 9 einfügen. Aus den vorgegebenen Zahlen leiten Sie die Lösung dabei durch Deduktion und Logik ab. Beispielsweise steht in dem obersten linken Feld des Sudoku-Rätsels aus Abbildung E-1 eine 5, sodass diese Zahl nicht ein weiteres Mal in der obersten Zeile, der linken Spalte oder dem 3x3-Quadrat oben links auftreten kann. Wenn Sie eine Zeile, eine Spalte oder ein Viereck nach dem anderen lösen, erhalten Sie weitere Hinweise für den Rest des Rätsels.

5	3		7					
6			1	9	5			
	9	8				6		
8			6				3	
4		8	3				1	
7			2			6		
	6				2	8		
		4	1	9			5	
		8			7	9		

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

**Abb. E-1** Ein Sudoku-Rätsel (links) und die Lösung (rechts). Beim Sudoku geht es zwar um Zahlen, doch ist dafür keine Mathematik erforderlich. (Bilder © Wikimedia Commons)

Nur weil es bei Sudoku um Zahlen geht, heißt das noch lange nicht, dass man gut in Mathe sein muss, um die Lösung auszuknöbeln. Das Gleiche gilt auch fürs Programmieren. Wie beim Sudoku müssen Sie auch beim Programmieren das Problem in einzelne Schritte zerlegen. Beim *Debuggen* von Programmen (also beim Aufspüren und Beheben von Fehlern) müssen Sie geduldig beobachten, was das Programm macht, und die Ursachen von Fehlern herausfinden. Wie bei allen anderen Fähigkeiten werden Sie auch beim Programmieren umso besser, je mehr Erfahrung Sie haben.

## Programmierung ist kreativ

Programmieren ist eine kreative Tätigkeit, ähnlich dem Bauen mit Lego-Steinen. Am Anfang haben Sie eine grobe Vorstellung von der Burg oder was auch immer Sie bauen möchten, und wissen, welche Steine Ihnen zur Verfügung stehen. Dann beginnen Sie zu bauen. Wenn Sie ein Programm geschrieben haben, können Sie den Code anschließend noch etwas verschönern – ebenso, wie Sie es bei der Lego-Burg tun würden.

Der Unterschied zwischen Programmierung und anderen kreativen Tätigkeiten besteht darin, dass Sie beim Programmieren das gesamte erforderliche Rohmaterial in Ihrem Computer haben. Sie müssen keine Leinwand, keine Farbe, keinen Film, kein Garn, keine Lego-Steine oder elektronischen Bauteile kaufen. Nachdem Sie das Programm geschrieben haben, können Sie es auf einfache Weise der ganzen Welt zur Verfügung stellen. Und obwohl Sie beim Programmieren Fehler begehen werden, macht die Sache doch viel Spaß.

## Der Aufbau dieses Buchs

Der erste Teil dieses Buchs behandelt die Grundlagen der Python-Programmierung. Im zweiten Teil sehen wir uns dann verschiedene Aufgaben an, die Sie automatisieren können. In jedem Kapitel des zweiten Teils gibt es Übungsprojekte. Die folgende Übersicht zeigt, was Sie in den einzelnen Kapiteln erwarten:

### Teil I: Grundlagen der Python-Programmierung

**Kapitel 1: Grundlagen von Python** Hier werden Ausdrücke vorgestellt, die grundlegendste Art von Python-Anweisungen. Außerdem erfahren Sie, wie Sie die interaktive Shell von Python verwenden, um Code auszuprobieren.

**Kapitel 2: Flusssteuerung** In diesem Kapitel erfahren Sie, wie Ihre Programme entscheiden können, welcher Code in einer bestimmten Situation ausgeführt werden soll. Dadurch können sie auf unterschiedliche Bedingungen reagieren.

**Kapitel 3: Funktionen** Dieses Kapitel zeigt Ihnen, wie Sie eigene Funktionen definieren, um Ihren Code in besser handhabbare Abschnitte zu gliedern.

**Kapitel 4: Listen** Hier erhalten Sie eine Einführung in den Datentyp der Listen und erfahren, wie Sie damit Daten gliedern können.

**Kapitel 5: Dictionaries und Datenstrukturen** Dieses Kapitel gibt eine Einführung in den Datentyp der Dictionaries und führt noch weitere Möglichkeiten auf, um Daten zu gliedern.

**Kapitel 6: Stringbearbeitung** Hier geht es um die Arbeit mit Textdaten (die in Python *Strings* genannt werden).

### Teil II: Aufgaben automatisieren

**Kapitel 7: Mustervergleich mit regulären Ausdrücken** Hier erfahren Sie, wie Sie mit regulären Ausdrücken nach Textmustern suchen können.

**Kapitel 8: Dateien lesen und schreiben** Dieses Kapitel erklärt, wie Ihre Programme den Inhalt von Textdateien lesen und selbst Informationen in Dateien auf der Festplatte speichern können.

**Kapitel 9: Dateien verwalten** Sie erfahren hier, wie Python große Mengen von Dateien kopieren, verschieben, umbenennen und löschen kann, und zwar viel schneller, als ein menschlicher Bearbeiter es tun könnte. Außerdem werden das Komprimieren und Entpacken von Dateien erklärt.

**Kapitel 10: Debugging** Hier werden die verschiedenen Instrumente vorgestellt, die in Python zur Verfügung stehen, um Fehler (Bugs) zu finden und zu beheben.

**Kapitel 11: Web Scraping** Dieses Kapitel zeigt Ihnen, wie Sie Programme schreiben, die automatisch Webseiten herunterladen und nach Informationen durchforsten. Dieser Vorgang wird *Web Scraping* genannt.

**Kapitel 12: Arbeiten mit Excel-Arbeitsblättern** Hier geht es darum, wie Sie Excel-Arbeitsblätter programmgesteuert bearbeiten, sodass Sie sie nicht selbst lesen müssen. Das ist besonders praktisch, wenn die Anzahl der Dokumente, die Sie analysieren müssen, in die Hunderte oder gar in die Tausende geht.

**Kapitel 13: Arbeiten mit PDF- und Word-Dokumenten** Dieses Kapitel behandelt das programmgesteuerte Lesen von Word- und PDF-Dokumenten.

**Kapitel 14: Arbeiten mit CSV-Dateien und JSON-Daten** Die Erklärung der programmgesteuerten Bearbeitung von Dokumenten wird hier anhand von CSV- und JSON-Dateien fortgesetzt.

**Kapitel 15: Zeit einhalten, Aufgaben zeitlich planen und Programme starten** Hier lernen Sie, wie Python Uhrzeiten und Kalenderdaten handhabt und Sie dafür sorgen, dass Ihr Computer Aufgaben zu einem bestimmten Zeitpunkt ausführt. Außerdem erfahren Sie, wie Sie von Python-Programmen aus andere Programme starten.

**Kapitel 16: E-Mails und Textnachrichten senden** In diesem Kapitel geht es darum, Programme zu schreiben, die an Ihrer Stelle E-Mails und Textnachrichten senden.

**Kapitel 17: Bildbearbeitung** Dieses Kapitel erklärt, wie Sie Bilder, z. B. JPEG- oder PNG-Dateien, in Ihren Programmen bearbeiten können.

**Kapitel 18: Tastatur und Maus mit GUI-Automatisierung steuern** Hier lernen Sie, wie Sie in Ihrem Programm die Maus und die Tastatur steuern, um Mausklicks und Tastenbetätigungen zu simulieren.

## Python herunterladen und installieren

Python können Sie kostenlos für Windows, OS X und Linux von <http://python.org/downloads/> herunterladen. Wenn Sie die neueste Version verwenden, die auf der Website angeboten wird, sollten alle Programme in diesem Buch funktionieren.

### Warnung

Achten Sie darauf, eine Version von Python 3 herunterzuladen (z. B. 3.4.0). Die Programme in diesem Buch sind für Python 3 geschrieben. Auf Python 2 funktionieren sie unter Umständen gar nicht oder zumindest nicht korrekt.

Auf der Download-Seite finden Sie Installer für die verschiedenen Betriebssysteme und dabei wiederum jeweils für 64- und für 32-Bit-Computer. Als Erstes müssen Sie daher herausfinden, welchen Installer Sie brauchen. Wenn Sie Ihren Computer 2007 oder später gekauft haben, handelt es sich sehr wahrscheinlich um ein 64-Bit-System, anderenfalls eher um einen 32-Bit-Rechner. Genau herausfinden können Sie das wie folgt:

- Auf Windows wählen Sie *Start > Systemsteuerung > System* und schauen nach, ob als Systemtyp 64 Bit oder 32 Bit angegeben wird.
- Auf OS X wählen Sie im Apfelmenü *Über diesen Mac > Weitere Informationen > Systembericht > Hardware*. Schauen Sie sich in der *Hardware-Übersicht* den Eintrag unter *Prozessortyp* an. Wenn dort *Intel Core Solo* oder *Intel Core Duo* steht, haben Sie einen 32-Bit-Rechner. Bei allen anderen Einträgen (auch *Intel Core 2 Duo*) handelt es sich um einen 64-Bit-Computer.
- Auf Ubuntu Linux geben Sie in einem Terminalfenster den Befehl `uname -m` ein. Die Antwort `i686` bedeutet, dass Sie einen 32-Bit-Computer haben. Bei einem 64-Bit-Rechner lautet die Antwort `x86_64`.

Laden Sie auf Windows den Python-Installer (mit der Endung *.msi*) herunter und doppelklicken Sie darauf. Befolgen Sie die Anweisungen, die auf dem Bildschirm angezeigt werden. Der Vorgang läuft wie folgt ab:

1. Wählen Sie *Install for All Users* und dann *Weiter*.
2. Akzeptieren Sie *C:\Python34* als Installationsordner, indem Sie auf *Weiter* klicken.
3. Klicken Sie erneut auf *Weiter*, um den Schritt *Customize Python* zu über-springen.

Auf Mac OS X laden Sie die passende *.dmg*-Datei für Ihre Version von OS X herunter und doppelklicken darauf. Befolgen Sie die Anweisungen, die auf dem Bildschirm angezeigt werden. Der Vorgang läuft wie folgt ab:

1. Wenn das DMG-Paket in einem neuen Fenster geöffnet wird, doppelklicken Sie auf die Datei *Python.mpkg*. Möglicherweise müssen Sie Ihr Administratortagswort eingeben.
2. Klicken Sie im gesamten Abschnitt *Welcome* auf *Continue* und auf *Agree*, um die Lizenzbedingungen zu akzeptieren.
3. Wählen Sie *HD Macintosh* (bzw. den Namen Ihrer Festplatte) aus und klicken Sie auf *Install*.

Auf Ubuntu können Sie Python wie folgt im Terminal installieren:

1. Öffnen Sie ein Terminalfenster.
2. Geben Sie `sudo apt-get install python3` ein.
3. Geben Sie `sudo apt-get install idle3` ein.
4. Geben Sie `sudo apt-get install python3-pip` ein.

## IDLE starten

Der *Python-Interpreter* ist die Software, die Ihre Python-Programme ausführt. Die Eingabe der Programme dagegen erfolgt in der *interaktiven Entwicklungsumgebung* (Interactive Development Environment, IDLE), ähnlich wie in einer Textverarbeitung. Um die IDLE zu starten, gehen Sie wie folgt vor:

- Auf Windows 7 und höher klicken Sie auf das Startsymbol in der linken unteren Ecke, geben IDLE in das Suchfeld ein und wählen *IDLE (Python GUI)* aus.
- Auf Windows XP klicken Sie auf die Startschaltfläche und wählen *Programme > Python 3.4 > IDLE (Python GUI)*.
- Auf Mac OS X öffnen Sie ein Finder-Fenster, klicken auf *Programme*, dann auf *Python 3.4* und schließlich auf das IDLE-Symbol.
- Auf Ubuntu wählen Sie *Anwendungen > Zubehör > Terminal* und geben dann **idle3** ein. (Möglichkeitweise können Sie auch oben auf dem Bildschirm auf *Anwendungen* klicken, dann *Programming* auswählen und auf *IDLE3* klicken.)

## Die interaktive Shell

Unabhängig vom Betriebssystem ist das IDLE-Fenster, das beim Start erscheint, bis auf folgenden Text größtenteils leer:

```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64
bit (AMD64)] on win32Type "copyright", "credits" or "license()" for more
information.
>>>
```

Dieses Fenster ist die *interaktive Shell*. Eine Shell ist ein Programm, in das Sie Anweisungen für den Computer eingeben können, ähnlich wie das Terminal von OS X oder die Befehlszeile von Windows. Die Anweisungen, die Sie in die interaktive Python-Shell eingeben, werden vom Python-Interpreter gelesen und sofort ausgeführt.

Probieren Sie das aus, indem Sie an der Eingabeaufforderung (`>>>`) der Shell Folgendes eingeben:

```
>>> print('Hello world!')
```

Wenn Sie nun die Eingabezeile drücken, zeigt die interaktive Shell die Reaktion an:

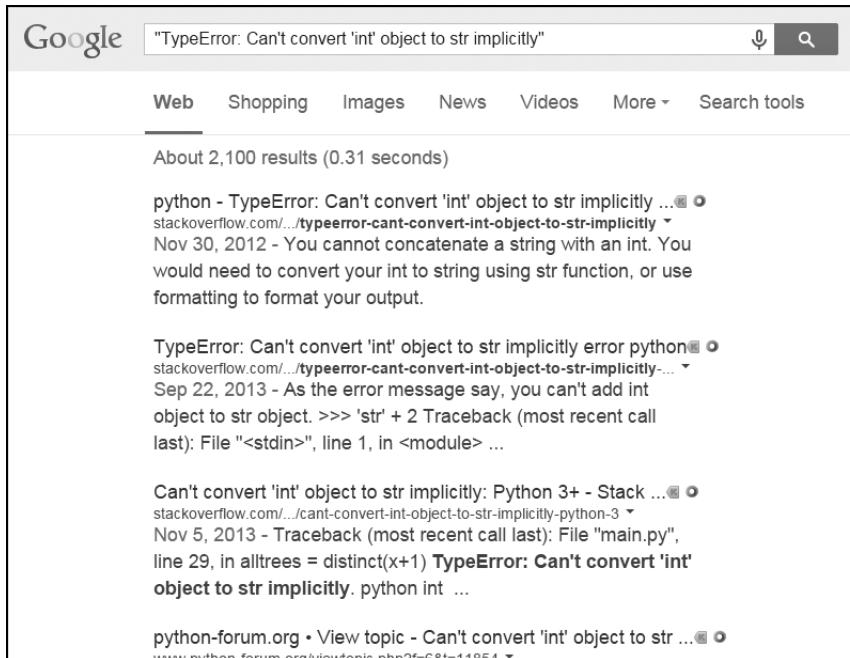
```
>>> print('Hello world!')
Hello world!
```

## Hilfe finden

Es ist viel einfacher, als Sie vielleicht glauben, Probleme bei der Programmierung selbst zu lösen. Sie glauben mir nicht? Dann provozieren wir einmal absichtlich einen Fehler: Geben Sie in die interaktive Shell `'42' + 3` ein. Machen Sie sich keine Gedanken darüber, was diese Anweisung bedeutet und was daran falsch sein soll, sondern achten Sie auf das Ergebnis:

```
>>> '42' + 3
Traceback (most recent call last): ❶
  File "<pyshell#0>", line 1, in <module>
    '42' + 3
TypeError: Can't convert 'int' object to str implicitly ❷
>>>
```

Da Python die Anweisung nicht versteht, erscheint hier eine Fehlermeldung (❷). Der als »Traceback« bezeichnete Teil der Fehlermeldung (❶) gibt die Anweisung und die Nummer der Zeile an, mit der Python Schwierigkeiten hat. Wenn Sie eine Fehlermeldung erhalten, die Ihnen schleierhaft ist, suchen Sie online nach der genauen Formulierung. In diesem Fall also würden Sie "TypeError: Can't convert 'int' object to str implicitly" (mit Anführungszeichen) in eine Suchmaschine eingeben. Daraufhin erhalten Sie jede Menge Links, in denen erklärt wird, was diese Fehlermeldung bedeutet und was die Ursache ist (siehe Abbildung E-2).



**Abb. E-2** Google-Ergebnisse für eine Fehlermeldung können sehr hilfreich sein.

Sie werden dabei sehr oft feststellen, dass jemand schon einmal die gleiche Frage hatte wie Sie und dass irgendeine hilfreiche Person sie bereits beantwortet hat. Niemand kann alles über Programmierung wissen. Zur täglichen Arbeit eines Softwareentwicklers gehört auch die Suche nach Antworten auf technische Fragen.

## Geschickte Fragen stellen

Wenn Sie die Antworten auf Ihre Fragen nicht durch eine Onlinesuche finden können, versuchen Sie, Teilnehmer in Webforen wie Stack Overflow (<http://stackoverflow.com/>) oder dem Subreddit »Learn Programming« auf <http://reddit.com/r/learnprogramming> zu fragen. Beachten Sie aber, dass Sie Ihre Fragen geschickt stellen müssen, damit andere Ihnen helfen können. Lesen Sie auf jeden Fall den FAQ-Abschnitt der Website, um zu erfahren, wie Sie Fragen auf richtige Weise vorbringen.

Wenn Sie Fragen zur Programmierung stellen, sollten Sie Folgendes tun:

- Erklären Sie nicht nur, was Sie getan haben, sondern auch, was Sie tun wollten. Dadurch können Helfer erkennen, ob Sie sich verrannt haben.
- Geben Sie genau an, wann der Fehler auftritt. Zeigt er sich gleich zu Beginn des Programms oder erst nach einer bestimmten Aktion?

- Kopieren Sie die gesamte Fehlermeldung und Ihren Code auf <http://pastebin.com/> oder <http://gist.github.com/>. Diese Websites erleichtern es, anderen Personen große Mengen an Code über das Web zur Verfügung zu stellen, ohne die Formatierung zu verlieren. Den URL zu dem dort veröffentlichten Code fügen Sie dann in Ihre E-Mail oder Ihren Forumspost ein. Als Beispiele können Sie sich Code von mir auf <http://pastebin.com/SzP2DbFx/> und [https://gist.github.com/asweigart/6912168/](https://gist.github.com/asweigart/6912168) ansehen.
- Erklären Sie, was Sie bereits versucht haben, um Ihr Problem zu lösen. Das zeigt den anderen, dass Sie selbst schon etwas Mühe darin investiert haben, die Lösung herauszufinden.
- Geben Sie an, welche Version von Python Sie verwenden. (Es gibt einige entscheidende Unterschiede zwischen den Python-Interpretern der Versionen 2 und 3.) Nennen Sie auch die Version Ihres Betriebssystems.
- Wenn ein Fehler nach einer Änderung am Code auftrat, erklären Sie, was Sie genau geändert haben.
- Geben Sie an, ob der Fehler jedes Mal auftritt, wenn Sie das Programm ausführen, oder nur, nachdem Sie bestimmte Aktionen durchgeführt haben. Beschreiben Sie in letzterem Fall auch diese Aktionen.

Befolgen Sie immer die Online-Etikette. Schreiben Sie also Ihre Posts nicht komplett in Großbuchstaben und stellen Sie keine unsinnigen Forderungen an die Menschen, die Ihnen zu helfen versuchen.

## Zusammenfassung

Für die meisten Menschen ist ein Computer eher ein Haushaltsgerät als ein Werkzeug. Wenn Sie jedoch zu programmieren lernen, steht Ihnen eines der vielseitigsten Werkzeuge der modernen Welt zur Verfügung und obendrein werden Sie auch noch Spaß dabei haben. Programmieren ist ganz anders als Gehirnchirurgie – Sie können sich auch als Anfänger daran versuchen, herumexperimentieren und gefahrlos Fehler machen.

Ich helfe gern anderen Menschen dabei, Python zu entdecken. In meinem Blog auf <http://inventwithpython.com/blog/> schreibe ich Programmertutorials. Wenn Sie Fragen haben, können Sie mich (in englischer Sprache) unter [al@inventwithpython.com](mailto:al@inventwithpython.com) erreichen.

Für dieses Buch müssen Sie keinerlei Programmierkenntnisse mitbringen. Es kann aber sein, dass Sie Fragen haben, die über den behandelten Stoff hinausgehen. Die richtigen Fragen zu stellen und zu wissen, wo Sie Antworten finden können, sind wertvolle Werkzeuge für die Programmierung.

Fangen wir an!

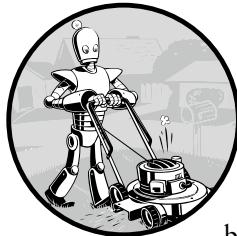
# Teil 1

**Grundlagen der Python-Programmierung**



# 1

## Grundlagen von Python



Die Programmiersprache Python verfügt über eine breite Palette von syntaktischen Konstruktionen, Standardbibliotheksfunktionen und Möglichkeiten zur interaktiven Entwicklung. Zum Glück brauchen Sie sich um das meiste davon nicht zu kümmern, sondern müssen nur so viel lernen, dass Sie damit praktische kleine Programme schreiben können.

Allerdings müssen Sie, bevor Sie irgendetwas tun können, zunächst einige Grundlagen der Programmierung erlernen. Wie ein Zauberlehrling werden Sie vielleicht denken, dass einige dieser Grundlagen ziemlich undurchsichtig sind und dass es viel Mühe macht, sie zu lernen, aber diese Kenntnisse und etwas Übung werden Sie in die Lage versetzen, Ihren Computer wie einen Zauberstab zu nutzen und damit unglaublich erscheinende Dinge zu tun.

In einigen Beispielen in diesem Kapitel werden Sie dazu aufgefordert, etwas in die interaktive Shell einzugeben. Damit können Sie einen Python-Befehl nach dem anderen ausführen und die Ergebnisse unmittelbar einsehen. Die Verwendung dieser Shell eignet sich hervorragend, um zu lernen, was die grundlegenden

Python-Anweisungen bewirken. Nutzen Sie sie daher, während Sie das Buch durcharbeiten. Auf diese Weise können Sie sich den Stoff besser merken, als wenn Sie ihn nur lesen würden.

## Ausdrücke in die interaktive Shell eingeben

Um die interaktive Shell auszuführen, müssen Sie die Entwicklungsumgebung IDLE starten, die Sie nach der Anleitung im Vorwort zusammen mit Python installiert haben. Unter Windows öffnen Sie das Startmenü und wählen *Alle Programme > Python 3.3 > IDLE (Python GUI)*. Unter OS X wählen Sie *Programme > MacPython 3.3 > Python*. Unter Ubuntu öffnen Sie ein neues Terminal-Fenster und geben `idle3` ein.

Daraufhin wird ein Fenster mit der Eingabeaufforderung `>>>` angezeigt. Dies ist die interaktive Shell. Geben Sie `2 + 2` ein, um Python eine einfache Berechnung ausführen zu lassen:

```
>>> 2 + 2
4
```

Das IDLE-Fenster sieht nun wie folgt aus:

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>>
```

In Python wird etwas wie `2 + 2` als *Ausdruck* bezeichnet. Dies ist die einfachste Form von Programmieranweisung in dieser Sprache. Ausdrücke setzen sich aus *Werten* (wie `2`) und *Operatoren* (wie `+`) zusammen. Sie können stets *ausgewertet*, also auf einen einzigen Wert reduziert werden. Dadurch können Sie an allen Stellen im Python-Code, an denen ein Wert stehen kann, auch einen Ausdruck verwenden.

Im vorstehenden Beispiel wurde `2 + 2` zu dem Wert `4` ausgewertet. Ein einzelner Wert ohne Operatoren wird ebenfalls als Ausdruck angesehen, wird aber nur zu sich selbst ausgewertet:

```
>>> 2
2
```

### Fehler sind kein Beinbruch

Wenn ein Programm Code enthält, den der Computer nicht versteht, stürzt es ab, woraufhin Python eine Fehlermeldung anzeigt. Ihren Computer können Sie dadurch jedoch nicht beschädigen. Daher brauchen Sie auch keine Angst vor Fehlern zu haben. Bei einem Absturz hält das Programm nur unerwartet an.

Wenn Sie mehr über eine bestimmte Fehlermeldung wissen wollen, können Sie online nach dem genauen Text suchen. Auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) finden Sie außerdem eine Liste häufig auftretender Python-Fehlermeldungen und ihrer Bedeutungen.

Es gibt eine Menge verschiedener Operatoren, die Sie in Python-Ausdrücken verwenden können. Tabelle 1–1 führt die arithmetischen Operatoren auf.

Operator	Operation	Beispiel	Ergebnis
<code>**</code>	Exponent	<code>2 ** 3</code>	8
<code>%</code>	Modulus/Rest	<code>22 % 8</code>	6
<code>//</code>	Integerdivision/abgerundeter Quotient	<code>22 // 8</code>	2
<code>/</code>	Division	<code>22 / 8</code>	2.75
<code>*</code>	Multiplikation	<code>3 * 5</code>	15
<code>-</code>	Subtraktion	<code>5 - 2</code>	3
<code>+</code>	Addition	<code>2 + 2</code>	4

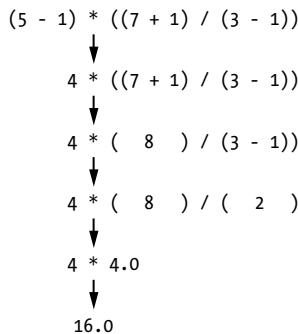
**Tab. 1–1** Arithmetische Operatoren, geordnet vom höchsten zum niedrigsten Vorrang

Die Auswertungsreihenfolge oder *Rangfolge* der arithmetischen Operatoren in Python entspricht ihrer gewöhnlichen Rangfolge in der Mathematik: Als Erstes wird der Operator `**` ausgewertet, dann die Operatoren `*`, `/`, `//` und `%` von links nach rechts und schließlich die Operatoren `+` und `-` (ebenfalls von links nach rechts). Um die Auswertungsreihenfolge zu ändern, können Sie bei Bedarf Klammern setzen. Zur Übung geben Sie die folgenden Ausdrücke in die interaktive Shell ein:

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
```

```
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2 + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

Die Ausdrücke müssen Sie jeweils selbst eingeben, aber Python nimmt Ihnen die Arbeit ab, sie auf einen einzelnen Wert zu reduzieren. Wie Abb. 1–1 zeigt, wertet es dabei nach und nach die einzelnen Teile eines Ausdrucks aus, bis ein einziger Wert übrig ist.



**Abb. 1–1** Durch die Auswertung wird ein Ausdruck auf einen einzigen Wert reduziert.

Die Regeln, nach denen Operatoren und Werte zu Ausdrücken zusammengestellt werden, bilden einen grundlegenden Bestandteil der Programmiersprache Python – ebenso wie die Grammatikregeln einer natürlichen Sprache. Betrachten Sie das folgende Beispiel:

*Dies ist ein grammatisch korrekter deutscher Satz.*

*Dies grammatisch ist Satz kein deutscher korrekter.*

Der zweite Satz lässt sich nur schwer verstehen (»parsen«, wie es bei einer Programmiersprache heißt), da er nicht den Regeln der deutschen Grammatik folgt. Genauso ist es, wenn Sie eine schlecht formulierte Python-Anweisung eingeben. Python versteht sie nicht und zeigt die Fehlermeldung `SyntaxError` an:

```
>>> 5 +
      File "<stdin>", line 1
      5 +
      ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
      File "<stdin>", line 1
      42 + 5 + * 2
      ^
SyntaxError: invalid syntax
```

Um herauszufinden, ob eine Anweisung funktioniert oder nicht, können Sie sie einfach in die interaktive Shell eingeben. Keine Angst, dadurch können Sie nichts kaputt machen: Schlimmstenfalls zeigt Python eine Fehlermeldung an. Für professionelle Softwareentwickler gehören Fehlermeldungen zum Alltag.

## Die Datentypen für ganze Zahlen, Fließkommazahlen und Strings

Ein *Datentyp* ist eine Kategorie für Werte, wobei jeder Wert zu genau einem Datentyp gehört. Die gebräuchlichsten Datentypen in Python finden Sie in Tabelle 1–2. Werte wie -2 und -30 sind beispielsweise *Integerwerte*. Dieser Datentyp (`int`) steht für ganze Zahlen. Zahlen mit Dezimalpunkt, z. B. 3.14, sind dagegen *Fließkommazahlen* und weisen den Typ `float` auf. Beachten Sie, dass ein Wert wie 42 ein Integer ist, 42.0 dagegen eine Fließkommazahl.

Datentyp	Beispiele
Integer	-2, -1, 0, 1, 2, 3, 4, 5
Fließkommazahlen	-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 cats'

**Tab. 1–2 Häufig verwendete Datentypen**

In Python-Programmen können auch Textwerte vorkommen, sogenannte *Strings* (`str`). Schließen Sie Strings immer in einfache Anführungszeichen ein (z. B. `'Hello'` oder `'Goodbye cruel world!'`), damit Python weiß, wo der String anfängt und wo er endet. Sie können sogar einen String erstellen, der gar keine Zeichen enthält, nämlich den *leeren String* `''`. In Kapitel 4 werden Strings ausführlicher behandelt.

Wenn Sie die Fehlermeldung `SyntaxError: EOL while scanning string literal` erhalten, haben Sie wahrscheinlich wie im folgenden Beispiel das schließende einfache Anführungszeichen am Ende eines Strings vergessen:

```
>>> 'Hello world!
SyntaxError: EOL while scanning string literal
```

## Stringverkettung und -wiederholung

Die Bedeutung eines Operators kann sich in Abhängigkeit von den Datentypen der Werte ändern, die rechts und links von ihm stehen. Beispielsweise fungiert `+` zwischen zwei Integer- oder Fließkommawerten als Additionsoperator, zwischen zwei Strings aber als *Stringverkettungsoperator*. Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

Dieser Ausdruck wird zu einem einzigen neuen String ausgewertet, der den Text der beiden Originalstrings enthält. Wenn Sie jedoch versuchen, den Operator `+` zwischen einem String und einem Integerwert einzusetzen, weiß Python nicht, wie es damit umgehen soll, und gibt eine Fehlermeldung aus:

```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    'Alice' + 42
TypeError: Can't convert 'int' object to str implicitly
```

Die Fehlermeldung `Can't convert 'int' object to str implicitly` bedeutet, dass Python glaubt, Sie wollten einen Integer mit dem String `'Alice'` verketten. Dazu aber müssten Sie den Integerwert ausdrücklich in einen String umwandeln, da Python dies nicht automatisch tun kann. (Die Umwandlung von Datentypen werden wir im Abschnitt »Analyse des Programms« weiter hinten in diesem Kapitel erklären und uns dabei mit den Funktionen `str()`, `int()` und `float()` beschäftigen.)

Zwischen zwei Integer- oder Fließkommawerten dient `*` als Multiplikationsoperator, doch zwischen einem String und einem Integerwert wird er zum *Stringwiederholungsoperator*. Um das auszuprobieren, geben Sie in die interaktive Shell Folgendes ein:

```
>>> 'Alice' * 5
'AppleAppleAppleApple'
```

Der Ausdruck wird zu einem einzigen Stringwert ausgewertet, der den ursprünglichen String so oft enthält, wie der Integerwert angibt. Die Stringwiederholung ist zwar ein nützlicher Trick, wird aber längst nicht so häufig angewendet wie die Stringverkettung.

Den Operator \* können Sie nur zwischen zwei numerischen Werten (zur Multiplikation) oder zwischen einem String- und einem Integerwert einsetzen (zur Stringwiederholung). In allen anderen Fällen zeigt Python eine Fehlermeldung an:

```
>>> 'Alice' * 'Bob'  
Traceback (most recent call last):  
  File "<pyshell#32>", line 1, in <module>  
    'Alice' * 'Bob'  
TypeError: can't multiply sequence by non-int of type 'str'  
>>> 'Alice' * 5.0  
Traceback (most recent call last):  
  File "<pyshell#33>", line 1, in <module>  

```

Es ist sinnvoll, dass Python solche Ausdrücke nicht auswertet. Schließlich ist es nicht möglich, zwei Wörter miteinander zu multiplizieren, und es dürfte auch ziemlich schwierig sein, einen willkürlichen String eine gebrochene Anzahl von Malen zu wiederholen.

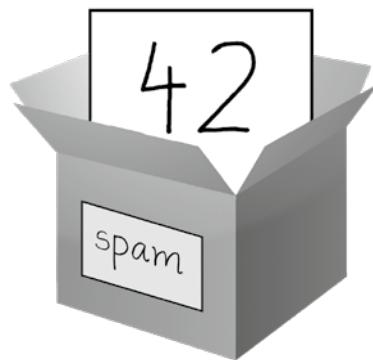
## Werte in Variablen speichern

Eine *Variable* können Sie sich wie eine Kiste im Arbeitsspeicher des Computers vorstellen, in der einzelne Werte abgelegt werden können. Wenn Sie das Ergebnis eines ausgewerteten Ausdrucks an späterer Stelle in Ihrem Programm noch brauchen, können Sie es in einer Variablen festhalten.

### Zuweisungsanweisungen

Um einen Wert in einer Variablen zu speichern, verwenden Sie eine *Zuweisungsanweisung*. Sie besteht aus einem Variablenamen, einem Gleichheitszeichen (das hier kein Gleichheitszeichen ist, sondern ein *Zuweisungsoperator*) und dem zu speichernden Wert. Wenn Sie die Zuweisungsanweisung `spam = 42` eingeben, wird der Wert 42 in der Variablen `spam` gespeichert.

Sie können sich eine Variable als eine beschriftete Kiste vorstellen, in der der Wert abgelegt wird (siehe Abb. 1–2).



**Abb. 1–2** Die Anweisung `spam = 42` sagt dem Programm: »Die Variable `spam` enthält jetzt die Ganzzahl 42.«

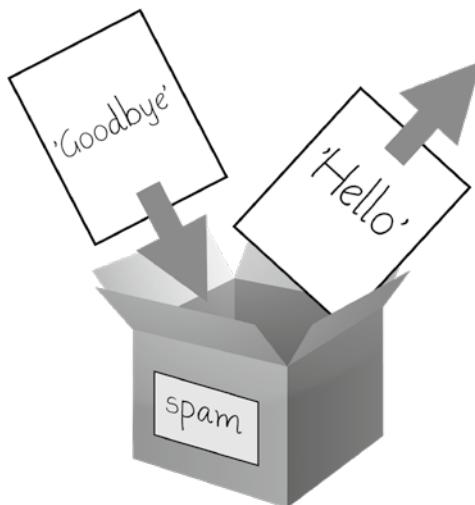
Geben Sie beispielsweise Folgendes in die interaktive Shell ein:

```
>>> spam = 40 ❶
>>> spam
40
>>> eggs = 2
>>> spam + eggs ❷
42
>>> spam + eggs + spam
82
>>> spam = spam + 2 ❸
>>> spam
42
```

Eine Variable wird *initialisiert* (erstellt), wenn zum ersten Mal ein Wert in ihr gespeichert wird (❶). Danach können Sie sie zusammen mit anderen Variablen und Werten in Ausdrücken verwenden (❷). Wenn Sie der Variablen einen neuen Wert zuweisen (❸), wird der alte Wert vergessen. Daher wird `spam` am Ende dieses Beispiels nicht mehr zu 40 ausgewertet, sondern zu 42. Die Variable ist also *überschrieben* worden. Versuchen Sie, in der interaktiven Shell wie folgt einen String zu überschreiben:

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

Wie die Kiste in Abb. 1–3 enthält die Variable `spam` in diesem Beispiel den Wert `Hello`, bis er durch `Goodbye` ersetzt wird.



**Abb. 1–3** Wird einer Variablen ein neuer Wert zugewiesen, so wird der alte vergessen.

## Variablennamen

Tabelle 1–3 führt Beispiele für gültige Variablennamen auf. Variablennamen müssen die folgenden drei Regeln erfüllen:

1. Der Name darf nur aus einem Wort bestehen.
2. Der Name darf nur aus Buchstaben, Ziffern und dem Unterstrich bestehen.
3. Der Name darf nicht mit einer Zahl beginnen.

Gültige Variablennamen	Ungültige Variablennamen
balance	current–balance (Bindestriche sind nicht zulässig)
currentBalance	current balance (Leerzeichen sind nicht zulässig)
current_balance	4account (der Name darf nicht mit einer Zahl beginnen)
_spam	42 (der Name darf nicht mit einer Zahl beginnen)
SPAM	total_ \$um (Sonderzeichen wie \$ sind nicht zulässig)
account4	'hello' (Sonderzeichen wie ' sind nicht zulässig)

**Tab. 1–3** Gültige und ungültige Variablennamen

Bei Variablennamen wird zwischen Groß- und Kleinschreibung unterschieden, sodass `spam`, `SPAM`, `Spam` und `sPaM` vier verschiedene Variablen bezeichnen. Verabredungsgemäß beginnen Variablen in Python mit einem Kleinbuchstaben.

In diesem Buch wird für Variablennamen die CamelCase-Schreibweise verwendet, also die Schreibung mit Binnenmajuskel statt mit einem Unterstrich. Variablennamen sehen also aus wie `lookLikeThis` und nicht wie `look_like_this`. Manche erfahrene Programmierer mögen einwenden, dass die offizielle Python-Stilrichtlinie PEP 8 Unterstriche verlangt. Ich bevorzuge allerdings die CamelCase-Schreibweise und möchte dazu auf den Abschnitt »Sinnlose Übereinstimmung ist die Plage kleiner Geister« aus PEP 8 verweisen:

*»Übereinstimmung mit der Stilrichtlinie ist wichtig. Am wichtigsten ist es jedoch zu wissen, wann man diese Übereinstimmung aufgeben muss. Für manche Fälle ist die Stilrichtlinie einfach ungeeignet. Urteilen Sie dann selbst nach bestem Wissen und Gewissen.«*

Ein guter Variablenname beschreibt, was für Daten darin festgehalten werden. Stellen Sie sich vor, Sie ziehen um und beschriften alle Kartons mit »Sachen«. Dann würden Sie Ihre Sachen nie wiederfinden! In diesem Buch und in einem Großteil der Python-Dokumentation werden allgemeine Variablennamen wie `spam`, `eggs` und `bacon` verwendet (in Anlehnung an den *Spam-Sketch* von Monty Python), aber in Ihren eigenen Programmen sollten Sie beschreibende Namen verwenden, um den Code leichter lesbar zu machen.

## Ihr erstes Programm

In der interaktiven Shell können Sie einzelne Python-Anweisungen nacheinander ausführen, aber um ein vollständiges Python-Programm zu schreiben, müssen Sie die Anweisungen in den *Dateieditor* eingeben. Er ähnelt Texteditoren wie dem Windows-Editor oder TextMate, verfügt aber zusätzlich über einige Sonderfunktionen für die Eingabe von Quellcode. Um den Dateieditor in IDLE zu öffnen, wählen Sie *File > New Window*.

In dem Fenster, das jetzt erscheint, sehen Sie einen Cursor, der auf Ihre Eingaben wartet. Dieses Fenster aber unterscheidet sich von der interaktiven Shell, in der Python-Anweisungen ausgeführt werden, sobald Sie die Eingabetaste drücken. Im Dateieditor können Sie viele Anweisungen eingeben, die Datei speichern und dann das Programm ausführen. Anhand der folgenden Merkmale können Sie erkennen, in welchem der beiden Fenster Sie sich gerade befinden:

- Das Fenster der interaktiven Shell zeigt die Eingabeaufforderung `>>>` an.
- Im Dateieditorfenster gibt es die Eingabeaufforderung `>>>` nicht.

Nun ist es an der Zeit, Ihr erstes Programm zu schreiben! Geben Sie im Fenster des Dateieditors Folgendes ein:

```
# Dieses Programm sagt "Hallo" und fragt nach Ihrem Namen. ❶
print('Hello world!') ❷
print('What is your name?') # Fragt nach dem Namen
myName = input() ❸
print('It is good to meet you, ' + myName) ❹
print('The length of your name is:') ❺
print(len(myName))
print('What is your age?') # Fragt nach dem Alter ❻
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Nachdem Sie den Quellcode eingegeben haben, speichern Sie ihn, damit Sie ihn nicht jedes Mal neu eingeben müssen, wenn Sie die IDLE starten. Wählen Sie im Menü oben im Editorfenster *File > Save As*, geben Sie im Speichern-Dialogfeld *hello.py* in das Feld *File Name* ein und klicken Sie auf *Save*.

Während Sie ein Programm eingeben, sollten Sie es zwischendurch immer mal wieder speichern. Sollte Ihr Computer abstürzen oder sollten Sie versehentlich die IDLE beenden, verlieren Sie dann keinen Code. Als Tastatatkürzel zum Speichern einer Datei drücken Sie **Strg** + **S** unter Windows und Linux bzw. **Cmd** + **S** unter OS X.

Nachdem Sie das Programm gespeichert haben, führen Sie es aus. Wählen Sie dazu *Run > Run Module* oder drücken Sie **F5**. Das Programm läuft jetzt im Fenster der interaktiven Shell, das sich beim Aufrufen der IDLE geöffnet hat. Beachten Sie aber, dass Sie **F5** im Fenster des Editorfensters drücken müssen, nicht im Shellfenster. Geben Sie Ihren Namen ein, wenn das Programm Sie danach fragt. Die Programmausgabe im Fenster der interaktiven Shell sieht wie folgt aus:

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello world!
What is your name?
A1
It is good to meet you, A1
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

Wenn keine Programmzeilen zum Ausführen mehr übrig sind, wird das Programm beendet.

Um den Dateieditor zu schließen, klicken Sie einfach auf das X in der rechten oberen Ecke. Wenn Sie ein gespeichertes Programm laden wollen, wählen Sie im Menü *File > Open*. Probieren Sie das jetzt aus. Wählen Sie in dem Fenster, das daraufhin erscheint, *hello.py* und klicken Sie auf *Open*. Jetzt erscheint wieder das zuvor gespeicherte Programm *hello.py* im Dateieditorfenster.

## Analyse des Programms

Wenn das neue Programm wieder im Dateieditor geöffnet ist, wollen wir uns genauer ansehen, was die einzelnen Programmzeilen bewirken und wozu die einzelnen Python-Anweisungen da sind.

### Kommentare

Die folgende Zeile ist ein sogenannter *Kommentar*:

```
# Dieses Programm sagt "Hallo" und fragt nach Ihrem Namen. ❶
```

Python ignoriert Kommentare. Sie können sie dazu verwenden, um Anmerkungen zu machen und sich selbst daran zu erinnern, was der Code tun soll. Der gesamte restliche Text der Zeile, die mit dem Zeichen # beginnt, gehört zum Kommentar.

Beim Testen von Programmen stellen Programmierer manchmal auch ein # vor eine Zeile mit Code, um sie vorübergehend zu entfernen. Diese Vorgehensweise wird *Auskommentieren* genannt. Das kann sehr hilfreich sein, wenn Sie herausfinden wollen, warum ein Programm nicht funktioniert. Wenn Sie später bereit sind, die Codezeile wieder in das Programm aufzunehmen, können Sie das # einfach entfernen.

Python ignoriert auch die Leerzeile unter dem Kommentar. Sie können in Ihr Programm so viele Leerzeilen einbauen, wie Sie wollen. Ähnlich wie Absätze in einem Buch machen sie den Code leichter lesbar.

### Die Funktion print()

Die Funktion `print()` gibt den Stringwert innerhalb der Klammern auf dem Bildschirm aus.

```
print('Hello world!') ❷  
print('What is your name?') # Fragt nach dem Namen
```

Die Zeile `print('Hello world!')` bedeutet: »Gib den Text des Strings 'Hello World!' aus.« Wenn Python diese Zeile ausführt, ruft Python die Funktion `print()` auf und übergibt ihr den Stringwert. Einen Wert, der an einem Funktionsaufruf übergeben wird, bezeichnet man als *Argument*. Beachten Sie, dass die Anführungszeichen nicht auf dem Bildschirm ausgegeben werden. Sie markieren nur den Anfang und das Ende des Strings, sind aber nicht Bestandteil des Stringwerts.

### Hinweis

Mit dieser Funktion können Sie auch eine leere Zeile auf dem Bildschirm ausgeben. Rufen Sie dazu einfach `print()` auf, also ohne irgendeinen Inhalt zwischen den Klammern.

Die öffnende und schließende Klammer am Ende eines Namens zeigen an, dass es sich um den Namen einer Funktion handelt. Aus diesem Grund sehen Sie in diesem Buch überall Bezeichnungen wie `print()` statt `print`. Funktionen werden in Kapitel 2 ausführlicher beschrieben.

### Die Funktion `input()`

Die Funktion `input()` wartet darauf, dass der Benutzer Text über die Tastatur eingibt und die Eingabetaste drückt.

```
myName = input() ❸
```

Dieser Funktionsaufruf wird zu dem String ausgewertet, den der Benutzer eingegeben hat. Die vorstehende Codezeile weist diesen Stringwert der Variablen `myName` zu.

Sie können sich den Funktionsaufruf `input()` als einen Ausdruck vorstellen, der zu dem String ausgewertet wird, den der Benutzer eingegeben hat. Gibt der Benutzer `A1` ein, so wird der Ausdruck zu `myName = 'A1'` ausgewertet.

### Den Benutzernamen ausgeben

Bei dem anschließenden Aufruf von `print()` steht in den Klammern der Ausdruck `'It is good to meet you, ' + myName'`:

```
print('It is good to meet you, ' + myName) ❹
```

Denken Sie daran, dass Ausdrücke immer zu einem einzigen Wert ausgewertet werden. Wenn in der vorherigen Zeile der Wert `'A1'` in `myName` gespeichert wurde, dann ergibt dieser Ausdruck `'It is good to meet you, A1'`. Dieser Stringwert wird nun an die Funktion `print()` übergeben, die ihn auf dem Bildschirm ausgibt.

## Die Funktion len()

Wenn Sie der Funktion `len()` einen Stringwert übergeben (oder eine Variable, die einen String enthält), wertet sie ihn zu einem Integer aus, der die Anzahl der Zeichen in diesem String angibt:

```
print('The length of your name is:')     ❸
print(len(myName))
```

Um das auszuprobieren, geben Sie Folgendes in die interaktive Shell ein:

```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
>>> len('')
0
```

Wie in diesen beiden Beispielen wird auch `len(myName)` zu einem Integer ausgewertet, der dann an `print()` übergeben wird, um ihn auf dem Bildschirm auszugeben. An `print()` lassen sich sowohl Integer- als auch Stringwerte übergeben. Schauen Sie sich aber die Fehlermeldung an, die sich ergibt, wenn Sie Folgendes in die interaktive Shell eingeben:

```
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print('I am ' + 29 + ' years old.')
TypeError: Can't convert 'int' object to str implicitly
```

Es ist nicht die Funktion `print()`, die den Fehler verursacht, sondern der Ausdruck, den Sie ihr zu übergeben versuchen. Dieselbe Fehlermeldung erhalten Sie auch, wenn Sie diesen Ausdruck ganz für sich allein in die Shell eingeben:

```
>>> 'I am ' + 29 + ' years old.'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'I am ' + 29 + ' years old.'
TypeError: Can't convert 'int' object to str implicitly
```

Python meldet einen Fehler, da der Operator `+` nur zwei Integer addieren oder zwei Strings verketten kann. Der Versuch, einen Integer und einen String zu verketten, widerspricht dagegen der Grammatik von Python. Diesen Fehler können Sie korrigieren, indem Sie eine Stringversion des Integers verwenden. Wie das geht, sehen wir uns im nächsten Abschnitt an.

## Die Funktionen str(), int() und float()

Wenn Sie eine Zahl wie 29 mit einem String verketten wollen, etwa um das Ergebnis an `print()` zu übergeben, brauchen Sie die Stringversion von 29, also '29'. Die Funktion `str()` nimmt einen Integer entgegen und wertet ihn zu seiner Stringversion aus:

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

Da `str(29)` den String '29' ergibt, wird der Ausdruck 'I am ' + `str(29)` + ' years old' zu 'I am ' + '29' + ' years old' ausgewertet und dieses wiederum zu 'I am 29 years old'. Dieser Wert wird dann an `print()` übergeben.

Die Funktionen `str()`, `int()` und `float()` werden zu der String-, Integer- bzw. Fließkommaversion des übergebenen Werts ausgewertet. Versuchen Sie in der aktiven Shell, einige Werte mithilfe dieser Funktionen umzuwandeln, und beobachten Sie, was passiert.

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

In den vorstehenden Beispielen werden die Funktionen `str()`, `int()` und `float()` aufgerufen und ihnen Werte anderer Datentypen übergeben, aus denen sie Strings, Integer- bzw. Fließkommazahlen machen.

Die Funktion `str()` ist insbesondere dann praktisch, wenn Sie eine Integer- oder Fließkommazahl haben, die Sie mit einem String verketten wollen. Liegt umgekehrt eine Zahl als Stringwert vor, dann können Sie die Funktion `int()` anwenden, um diese Zahl in mathematischen Funktionen einsetzen zu können. Das ist beispielsweise bei der Verwendung der Funktion `input()` wichtig, die stets einen

String zurückgibt, auch wenn der Benutzer eine Zahl eingibt. Geben Sie in der interaktiven Shell `spam = input()` ein. Wenn die Shell auf Ihren Text wartet, schreiben Sie **101**. Dabei geschieht Folgendes:

```
>>> spam = input()
101
>>> spam
'101'
```

In `spam` ist nicht etwa der Integer `101` gespeichert, sondern der String `'101'`. Wenn Sie mit dem Wert dieser Variablen nun irgendwelche Berechnungen anstellen wollen, müssen sie ihn zunächst mit `int()` in die Integerform umwandeln und diese als neuen Wert in `spam` speichern.

```
>>> spam = int(spam)
>>> spam
101
```

Jetzt können Sie die Variable `spam` wie einen Integer verwenden und nicht mehr wie einen String:

```
>>> spam * 10 / 5
202.0
```

Wenn Sie `int()` einen Wert übergeben, der nicht in einen Integer umgewandelt werden kann, zeigt Python eine Fehlermeldung an.

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

Mit der Funktion `int()` können Sie auch einen Fließkommawert abrunden.

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

In Ihrem ersten Programm werden die Funktionen `int()` und `str()` in den letzten drei Zeilen verwendet, um im Code Werte des passenden Datentyps bereitzustellen.

```
print('What is your age?')      # Fragt nach dem Alter ❻
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Die Variable `myAge` enthält den von `input()` zurückgegebenen Wert. Da diese Funktion immer einen String zurückgibt (auch wenn der Benutzer eine Zahl eingegeben hat), müssen Sie den String in `myAge` mit `int(myAge)` in einen Integerwert umwandeln, damit Sie 1 addieren können, was in dem Ausdruck `int(myAge) + 1` geschieht.

Das Ergebnis dieser Addition wird wiederum der Funktion `str()` übergeben: `str(int(myAge) + 1)`. Der daraus resultierende Stringwert wird mit den Strings '`You will be`' und '`in a year`' zu einem einzigen, langen Stringwert verkettet, der dann schließlich zur Anzeige an `print()` übergeben wird.

Nehmen wir an, der Benutzer gibt als Alter 4 ein. Der String '4' wird in einen Integer umgewandelt, sodass 1 addiert werden kann, was 5 ergibt. Die Funktion `str()` konvertiert dieses Ergebnis wiederum zu einem String zurück, sodass er mit dem zweiten String '`in a year`' verkettet werden kann, um die endgültige Meldung zu formen. Die Auswertung läuft wie in Abb. 1–4 gezeigt ab.

### Gleichheit von Text und Zahlen

Stringwerte sind etwas völlig anderes als Integer- und Fließkommawerte, allerdings können zwischen Integer- und Fließkommawerten Gleichheitsbeziehungen bestehen.

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

Python macht diese Unterscheidung, da Strings Text darstellen, Integer- und Fließkommawerte aber Zahlen.

```

    print('You will be ' + str(int(myAge) + 1) + ' in a year.')
    print('You will be ' + str(int( '4' ) + 1) + ' in a year.')
    print('You will be ' + str(      4 + 1      ) + ' in a year.')
    print('You will be ' +      5      ) + ' in a year.')
    print('You will be ' +          '5'      + ' in a year.')
    print('You will be 5'           + ' in a year.')
    print('You will be 5 in a year.')

```

**Abb. 1–4** Die Auswertungsschritte bei der Speicherung des Werts 4 in myAge

## Zusammenfassung

Sie können arithmetische Ausdrücke mit einem Taschenrechner berechnen und Strings in einer Textverarbeitung verketten. Durch Kopieren und Einfügen erreichen Sie sogar auf ganz einfache Weise eine Stringwiederholung. Ausdrücke mit ihren Komponenten wie Operatoren, Variablen und Funktionsaufrufen sind dagegen die Grundbausteine von Programmen. Wenn Sie mit diesen Elementen umgehen können, sind Sie in der Lage, Python anzuweisen, große Datenmengen für Sie zu verarbeiten.

Was Sie sich auf jeden Fall merken sollten, sind die verschiedenen Arten von Operatoren (die arithmetischen Operatoren `+`, `-`, `*`, `/`, `//`, `%` und `**` sowie die Stringoperatoren `+` und `*`) und die drei in diesem Kapitel vorgestellten Datentypen (Integer, Fließkommawerte und Strings).

Sie haben auch schon einige Funktionen gelernt. `print()` kümmert sich um die einfache Textausgabe (auf dem Bildschirm), `input()` um die Eingabe (von der Tastatur). Die Funktion `len()` nimmt einen String entgegen und wertet ihn zu einem Integer aus, der die Anzahl der Zeichen in dem String wiedergibt. Mit `str()`, `int()` und `float()` ermitteln Sie die String-, Integer- bzw. Fließkommaversion des übergebenen Werts.

Im nächsten Kapitel lernen Sie, wie Sie in Python auf der Grundlage eines vorliegenden Werts entscheiden, welcher Code ausgeführt oder übersprungen oder wiederholt werden soll. Diese Vorgehensweise wird als *Flusssteuerung* bezeichnet. Damit können Sie Programme schreiben, die intelligente Entscheidungen treffen.

## Wiederholungsfragen

1. Welche der folgenden Einträge sind Operatoren und welche sind Werte?

\*  
'hello'  
-88.8  
-  
/  
+  
5

2. Welcher der folgenden Einträge ist eine Variable und welcher ein String?

spam  
'spam'

3. Nennen Sie drei Datentypen!
4. Woraus besteht ein Ausdruck? Was machen alle Ausdrücke?
5. In diesem Kapitel wurden Zuweisungsanweisungen wie spam = 10 vorgestellt. Was ist der Unterschied zwischen einem Ausdruck und einer Anweisung?
6. Welchen Wert enthält die Variable bacon, nachdem der folgende Code ausgeführt worden ist?

bacon = 20  
bacon + 1

7. Wozu werden die beiden folgenden Ausdrücke ausgewertet?

'spam' + 'spamspam'  
'spam' \* 3

8. Warum ist eggs ein gültiger Variablenname, 100 dagegen nicht?
9. Mit welchen drei Funktionen können Sie die Integer-, Fließkomma- oder Stringversion eines Werts ermitteln?
10. Warum ruft der folgende Ausdruck eine Fehlermeldung hervor? Wie können Sie ihn korrigieren?

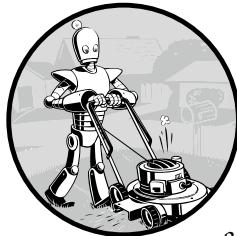
'I have eaten ' + 99 + ' burritos.'

**Zusatzpunkt:** Suchen Sie online nach der Python-Dokumentation für die Funktion len(). Sie befindet sich auf einer Webseite mit dem Titel »Built-in Functions«. Schauen Sie sich in der Liste weitere Funktionen von Python an, schlagen Sie nach, was die Funktion round() macht, und experimentieren Sie damit in der interaktiven Shell.



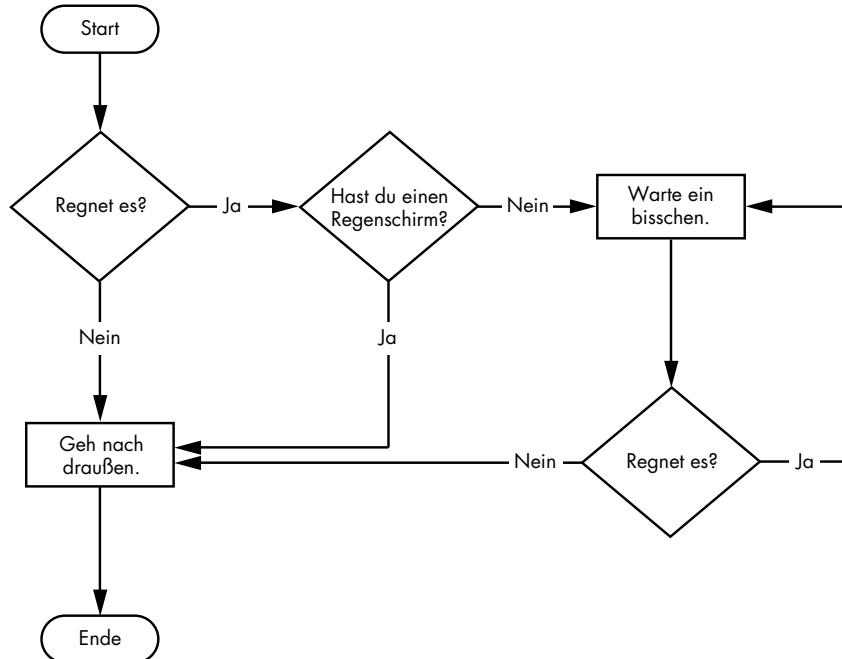
# 2

## Flusssteuerung



Sie kennen jetzt die Grundlagen für einzelne Anweisungen und wissen, dass ein Programm nichts anderes als eine Abfolge von Anweisungen ist. Die wahre Stärke der Programmierung besteht aber nicht darin, einfach nur eine Anweisung nach der anderen *auszuführen*, etwa so, wie Sie eine Einkaufsliste abarbeiten. Je nach dem Ergebnis, das die Auswertung eines Ausdrucks hat, kann das Programm entscheiden, Anweisungen zu überspringen, zu wiederholen oder unter mehreren möglichen Anweisungen auszuwählen. In der Praxis wird ein Programm so gut wie nie von der ersten bis zur letzten Anweisung Zeile für Zeile in gerader Linie ausgeführt. Mithilfe von *Flusssteuerungsanweisungen* wird entschieden, welche Anweisungen unter welchen Bedingungen auszuführen sind.

Da diese Flusssteuerungsanweisungen unmittelbar den Symbolen in einem Flussdiagramm entsprechen, zeige ich Ihnen in diesem Kapitel auch immer das Flussdiagramm zu dem besprochenen Code. Zur Einführung enthält Abb. 2–1 das Flussdiagramm, um zu entscheiden, was zu tun ist, wenn es regnet. Folgen Sie dem Pfad entlang der Pfeile vom Anfang bis zum Ende.



**Abb. 2-1** Ein Flussdiagramm, das Ihnen sagt, was Sie tun müssen, wenn es regnet.

In einem Flussdiagramm gibt es gewöhnlich mehr als eine Möglichkeit, um vom Start zum Ende zu gelangen. Das Gleiche gilt auch für die Codezeilen in einem Computerprogramm. In Flussdiagrammen werden diese Verzweigungen durch Rauten dargestellt. Für die anderen Schritte werden Rechtecke genommen, für Anfang und Ende abgerundete Rechtecke.

Bevor Sie die Flusssteuerungsanweisungen kennenlernen, müssen Sie zunächst einmal wissen, wie Sie die Optionen *ja* und *nein* darstellen und wie Sie die Verzweigungspunkte als Python-Code schreiben. Dazu beschäftigen wir uns mit booleschen Werten, Vergleichsoperatoren und booleschen Operatoren.

## Boolesche Werte

Für die Integer-, Fließkomma- und Stringdatentypen gibt es eine unendliche Anzahl an möglichen Werten, doch für den *booleschen* Datentyp (benannt nach dem Mathematiker George Boole) nur zwei, nämlich wahr und falsch (`True` und `False`). In Python-Code werden die booleschen Werte `True` und `False` immer ohne die Anführungszeichen für Strings und immer mit großem Anfangsbuchstaben geschrieben. Der Rest des Worts steht jeweils in Kleinbuchstaben. Geben Sie zum Ausprobieren folgenden Code in die interaktive Shell ein (wobei einige dieser Anweisungen absichtlich nicht korrekt sind und Fehlermeldungen hervorrufen):

```
>>> spam = True ❶
>>> spam
True
>>> true ❷
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    true
NameError: name 'true' is not defined
>>> True = 2 + 2 ❸
SyntaxError: assignment to keyword
```

Wie alle anderen Werte können auch boolesche Werte in Ausdrücken verwendet und in Variablen gespeichert werden (❶). Wenn Sie Groß- und Kleinschreibung verwechseln (❷) oder wenn Sie versuchen, True oder False als Variablennamen zu verwenden (❸), gibt Python eine Fehlermeldung aus.

## Vergleichsoperatoren

*Vergleichsoperatoren* vergleichen zwei Werte, wobei das Ergebnis ein einzelner boolescher Wert ist. Tabelle 2–1 führt die möglichen Vergleichsoperatoren auf.

Operator	Bedeutung
==	Gleich
!=	Ungleich
<	Kleiner als
>	Größer als
<=	Kleiner oder gleich
>=	Größer oder gleich

**Tab. 2–1** Vergleichsoperatoren

Je nachdem, welche Werte Sie übergeben, werden diese Operatoren zu True oder False ausgewertet. Im Folgenden wollen wir einige Operatoren ausprobieren, wobei wir mit == und != beginnen.

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

Wie zu erwarten ist, wird `==` (gleich) zu `True` ausgewertet, wenn die Werte auf beiden Seiten gleich sind, `!=` (ungleich) dagegen, wenn sie verschieden sind. Die Operatoren `==` und `!=` können für Werte beliebiger Datentypen verwendet werden.

```
>>> 'hello' == 'hello'  
True  
>>> 'hello' == 'Hello'  
False  
>>> 'dog' != 'cat'  
True  
>>> True == True  
True  
>>> True != False  
True  
>>> 42 == 42.0  
True  
>>> 42 == '42' ❶  
False
```

Beachten Sie, dass Integer- und Fließkommawerte immer ungleich Stringwerten sind. Der Ausdruck `42 == '42'` (❶) wird zu `False` ausgewertet, da für Python der Integer `42` und der String `'42'` zwei verschiedene Dinge sind.

Die Operatoren `<`, `>`, `<=` und `>=` dagegen funktionieren nur bei Integer- und Fließkommawerten.

```
>>> 42 < 100  
True  
>>> 42 > 100  
False  
>>> 42 < 42  
False  
>>> eggCount = 42  
>>> eggCount <= 42 ❷  
True  
>>> myAge = 29  
>>> myAge >= 10 ❸  
True
```

### Der Unterschied zwischen den Operatoren == und =

Wahrscheinlich ist Ihnen schon aufgefallen, dass der Gleichheitsoperator == aus zwei Gleichheitszeichen besteht, der Zuweisungsoperator = dagegen nur aus einem. Diese beiden Operatoren lassen sich sehr leicht verwechseln. Merken Sie sich einfach Folgendes:

- Mit dem Operator == (Gleichheit) fragen Sie, ob zwei Werte gleich sind.
- Mit dem Operator = (Zuweisung) weisen Sie den Wert auf der rechten Seite der Variablen auf der linken Seite zu.

Um sich zu merken, welcher dieser Operatoren welcher ist, denken Sie daran, dass der Gleichheitsoperator == ebenso wie der Ungleichheitsoperator != aus zwei Zeichen besteht.

Häufig verwenden Sie Vergleichsoperatoren, um einen Wert mit dem Wert zu vergleichen, der in einer Variablen gespeichert ist, so wie in den Beispielen eggCount <= 42 (❶) und myAge >= 10 (❷). (Schließlich brauchen Sie so etwas wie 'dog' != 'cat' gar nicht ausdrücklich in Ihrem Code zu schreiben, Sie könnten stattdessen auch gleich True sagen.) Weitere Beispiele dazu werden Sie sehen, wenn wir uns mit Flusssteuerungsstrukturen beschäftigen.

## Boolesche Operatoren

Die drei booleschen Operatoren and, or und not dienen dazu, boolesche Werte zu vergleichen. Wie Vergleichsoperatoren werten auch sie die betreffenden Ausdrücke zu einem einzigen booleschen Wert aus. Sehen wir uns diese Operatoren nun etwas genauer an. Dabei wollen wir mit dem Operator and beginnen.

### Binäre boolesche Operatoren

Da and und or stets zwei boolesche Werte (oder Ausdrücke) entgegennehmen, werden sie als *binäre* Operatoren bezeichnet. Dabei wird ein Ausdruck mit and zu True ausgewertet, wenn *beide* booleschen Werte True sind; anderenfalls ist der Ausdruck False. Um das auszuprobieren, geben Sie einige Ausdrücke mit and in die interaktive Shell ein:

```
>>> True and True  
True  
>>> True and False  
False
```

Eine *Wahrheitswertetafel* zeigt alle möglichen Ergebnisse eines booleschen Operators. In Tabelle 2–2 sehen Sie die Wahrheitswertetafel für den Operator `and`.

Ausdruck	Ergebnis
True and True	True
True and False	False
False and True	False
False and False	False

**Tab. 2–2** Wahrheitswertetafel für den Operator `and`

Ein Ausdruck mit dem Operator `or` dagegen wird zu `True` ausgewertet, wenn *mindestens einer* der beiden booleschen Werte `True` ist. Nur wenn beide `False` sind, ist der Ausdruck `False`

```
>>> False or True
True
>>> False or False
False
```

Alle möglichen Ergebnisse des Operators `or` sehen Sie in der Wahrheitswertetafel in Tabelle 2–3.

Ausdruck	Ergebnis
True or True	True
True or False	True
False or True	True
False or False	False

**Tab. 2–3** Wahrheitswertetafel für den Operator `or`

### Der Operator `not`

Anders als `and` und `or` wird der Operator `not` nur auf einen einzelnen booleschen Wert (oder einen Ausdruck) angewendet. Das Ergebnis ist einfach der gegenteilige boolesche Wert.

```
>>> not True
False
>>> not not not not True ❶
True
```

Ebenso wie bei doppelten Verneinungen in Schrift und Sprache können Sie auf not-Operatoren verschachteln (❶), allerdings gibt es keinen Grund dafür, so etwas in einem Programm in der Praxis zu tun. Tabelle 2–4 zeigt die Wahrheitswertetafel für not.

Ausdruck	Ergebnis
not True	False
not False	True

**Tab. 2–4** Wahrheitswertetafel für den Operator not

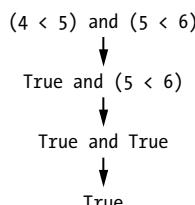
## Kombinierte Verwendung von booleschen und Vergleichsoperatoren

Da Vergleichsoperatoren zu booleschen Werten ausgewertet werden, können Sie sie in Ausdrücken mit booleschen Operatoren verwenden.

Denken Sie daran, dass and, or und not als boolesche Operatoren bezeichnet werden, da sie immer auf den booleschen Werten True und False operieren. Ausdrücke wie  $4 < 5$  sind zwar selbst keine booleschen Werte, werden aber zu booleschen Werten ausgewertet. Versuchen Sie, einige boolesche Ausdrücke mit Vergleichsoperatoren in die interaktive Shell einzugeben:

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

Der Computer wertet als Erstes den linken und dann den rechten Ausdruck aus. Wenn er die booleschen Werte für beide Ausdrücke kennt, wertet er den Gesamt- ausdruck zu einem einzigen booleschen Wert aus. Den Auswertungsvorgang für  $(4 < 5)$  and  $(5 < 6)$  sehen Sie in Abb. 2–2



**Abb. 2–2** Die Auswertung von  $(4 < 5)$  and  $(5 < 6)$  zu True

In einem Ausdruck können Sie auch mehrere boolesche sowie Vergleichsoperatoren verwenden:

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

Ebenso wie für arithmetische Operatoren gibt es auch eine Verarbeitungsreihenfolge für boolesche Operatoren. Nachdem alle arithmetischen und Vergleichsoperatoren ausgewertet wurden, verarbeitet Python als Erstes `not`, danach die `and`- und schließlich die `or`-Operatoren.

## Elemente zur Flusssteuerung

Die meisten Flusssteuerungsanweisungen beginnen mit einer *Bedingung*, auf die ein Codeblock folgt, die sogenannte *Klausel*. Bevor Sie die einzelnen Flusssteuerungsanweisungen in Python kennenlernen, sehen wir uns an, was Bedingungen und Codeblöcke eigentlich sind.

### Bedingungen

Alle bisher vorgestellten booleschen Ausdrücke können auch als Bedingungen bezeichnet werden. Eine Bedingung ist eine Art von Ausdruck im Zusammenhang mit Flusssteuerungsanweisungen. Bedingungen werden immer zu einem booleschen Wert ausgewertet, also zu `True` oder `False`, auf deren Grundlage die Flusssteuerungsanweisung dann entscheidet, was zu tun ist. Fast alle Flusssteuerungsanweisungen greifen auf eine Bedingung zurück.

### Codeblöcke

In Python können Codezeilen zu *Blöcken* gruppiert werden. Wo ein Block anfängt und wo er endet, können Sie an den Einrückungen der Codezeilen erkennen. Es gibt die folgenden drei Regeln für Blöcke:

1. Ein Block beginnt, wenn die Codezeilen weiter eingerückt werden.
2. Blöcke können andere Blöcke einschließen.
3. Ein Block endet, wenn die Einrückung auf null oder auf die Einrückung des einschließenden Blocks zurückgeht.

Das Prinzip von Blöcken lässt sich leichter verstehen, wenn Sie sich eingerückten Code ansehen. Suchen Sie Blöcke in diesem Programmausschnitt:

```
if name == 'Mary':  
    print('Hello Mary')    ❶  
    if password == 'swordfish':  
        print('Access granted.') ❷  
    else:  
        print('Wrong password.') ❸
```

Der erste Codeblock (❶) beginnt mit der Zeile `print('Hello Mary')` und enthält alle darauf folgenden Codezeilen. In diesem Block gibt es einen weiteren Block (❷), der nur aus einer einzigen Zeile besteht, nämlich `print('Access Granted.')`. Der dritte Block (❸) ist ebenfalls nur eine Zeile lang: `print('Wrong password.')`.

## Programmausführung

Im Programm `hello.py` aus dem vorigen Kapitel hat Python mit der Ausführung der Anweisungen am Anfang des Programms begonnen und sich dann eine Anweisung nach der anderen vorgearbeitet. Wenn Sie den Quellcode auf Papier ausgeben und mit dem Finger den Zeilen folgen, während sie abgearbeitet werden, markiert Ihr Finger jeweils die Stelle, bei der die Programmausführung gerade angekommen ist.

Nicht alle Programme werden jedoch einfach dadurch ausgeführt, dass die Anweisungen in einer Reihe nacheinander abgearbeitet werden. Wenn Sie mit Ihrem Finger ein Programm nachvollziehen, das Flusssteuerungsanweisungen enthält, werden Sie auf der Grundlage der vorliegenden Bedingungen hierhin und dorthin springen und ganze Klauseln auslassen.

## Flusssteuerungsanweisungen

Sehen wir uns nun die wichtigsten Bestandteile der Flusssteuerung an, nämlich die Anweisungen dafür. Sie entsprechen den Rauten in dem Flussdiagramm aus Abb. 2–1 und sind die Stellen, an denen das Programm Entscheidungen trifft.

### If-Anweisungen

Die am häufigsten vorkommende Art von Flusssteuerungsanweisungen ist die `if`-Anweisung. Ihre Klausel (also der Block, der auf die Bedingung folgt) wird ausgeführt, wenn ihre Bedingung `True` ist, anderenfalls wird sie übersprungen.

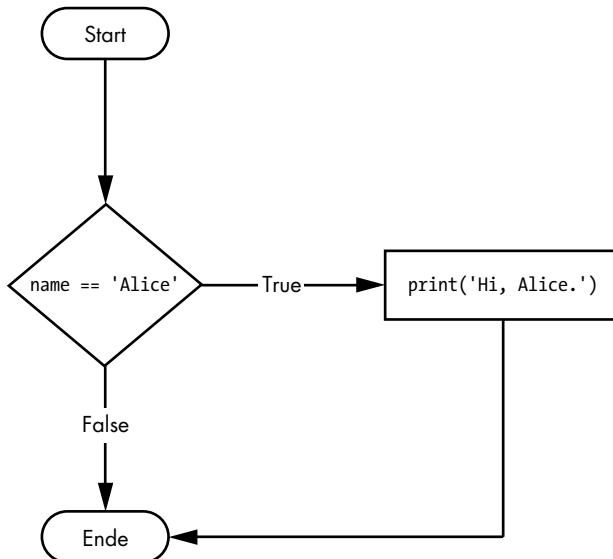
In Alltagssprache kann eine `if`-Anweisung wie folgt formuliert werden: »Wenn diese Bedingung wahr ist, dann führe den Code in der Klausel aus.« In Python besteht eine `if`-Anweisung aus folgenden Elementen:

- Dem Schlüsselwort `if`
- Einer Bedingung (ein Ausdruck, der zu `True` oder `False` ausgewertet wird)
- Einen Doppelpunkt
- Einen eingerückten Codeblock (die `if`-Klausel), der in der nächsten Zeile beginnt

Beispielsweise soll der folgende Code prüfen, ob der Name einer Person Alice ist (wobei Sie voraussetzen müssen, dass der Variablen `name` zuvor ein Wert zugewiesen wurde):

```
if name == 'Alice':
    print('Hi, Alice.')
```

Alle Flusssteuerungsanweisungen enden mit einem Doppelpunkt, auf den ein neuer Codeblock (die Klausel) folgt. In unserem Beispiel ist dies der Block `print('Hi, Alice.')`. In Abb. 2–3 sehen Sie das Flussdiagramm zu diesem Code.



**Abb. 2–3** Flussdiagramm einer `if`-Anweisung

### Else-Anweisungen

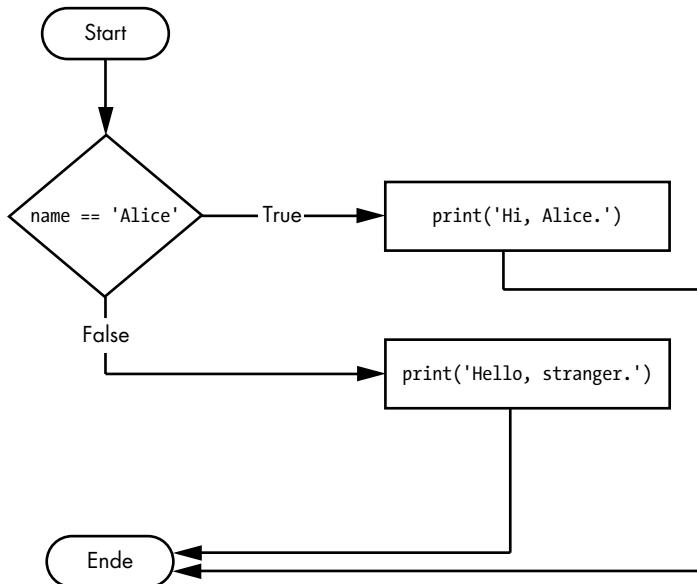
Auf eine `if`-Klausel kann optional auch eine `else`-Anweisung folgen. Die `else`-Klausel wird nur dann ausgeführt, wenn die Bedingung der `if`-Anweisung `False` ist. Eine `else`-Anweisung entspricht also folgender Formulierung: »Wenn diese Bedingung wahr ist, führe diesen Code aus, wenn nicht, dann jenen Code.« Die `else`-Anweisung braucht keine Bedingung. Sie besteht immer aus:

- Dem Schlüsselwort `else`
- Einem Doppelpunkt
- Einem eingerückten Codeblock (die `else`-Klausel), der in der nächsten Zeile beginnt

In unserem Alice-Beispiel können wir eine `else`-Anweisung verwenden, um eine andere Begrüßung anzuzeigen, wenn die Person nicht den Namen Alice hat:

```
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

Das Flussdiagramm für diesen Code sehen Sie in Abb. 2–4.



**Abb. 2–4** Flussdiagramm einer `else`-Anweisung

## Elif-Anweisungen

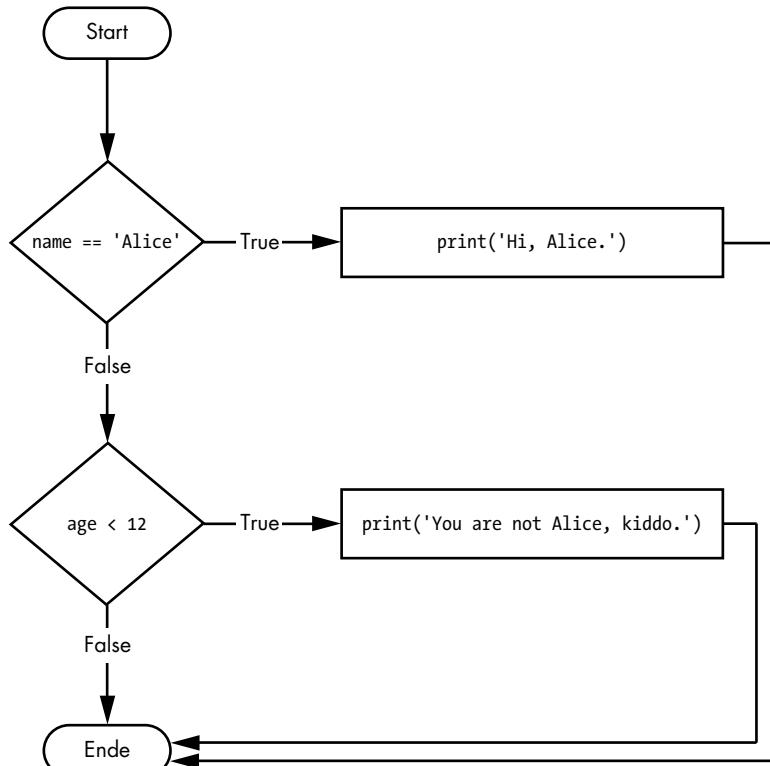
Bei den bisherigen Anweisungen wird entweder die `if`- oder die `else`-Klausel ausgeführt, doch es kann auch sein, dass Sie eine von *vielen* möglichen Klauseln ausführen lassen wollen. Die `elif`-Anweisung (»`else if`«) kann im Anschluss an eine `if`- oder an eine andere `elif`-Anweisung eingesetzt werden, damit eine weitere Bedingung geprüft wird, falls die vorherigen alle `False` waren. Eine `elif`-Anweisung weist immer folgende Bestandteile auf:

- Das Schlüsselwort `elif`
- Eine Bedingung (ein Ausdruck, der zu `True` oder `False` ausgewertet wird)
- Einen Doppelpunkt
- Einen eingerückten Codeblock (die `elif`-Klausel), der in der nächsten Zeile beginnt

Um die `elif`-Anweisung in Aktion zu erleben, fügen wir sie unserem Namensprüfungsprogramm hinzu:

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

Dieses Mal wird das Alter der Person geprüft und wenn sie jünger als 12 ist, teilt das Programm ihr etwas anderes mit. Das Flussdiagramm dafür sehen Sie in Abb. 2–5.



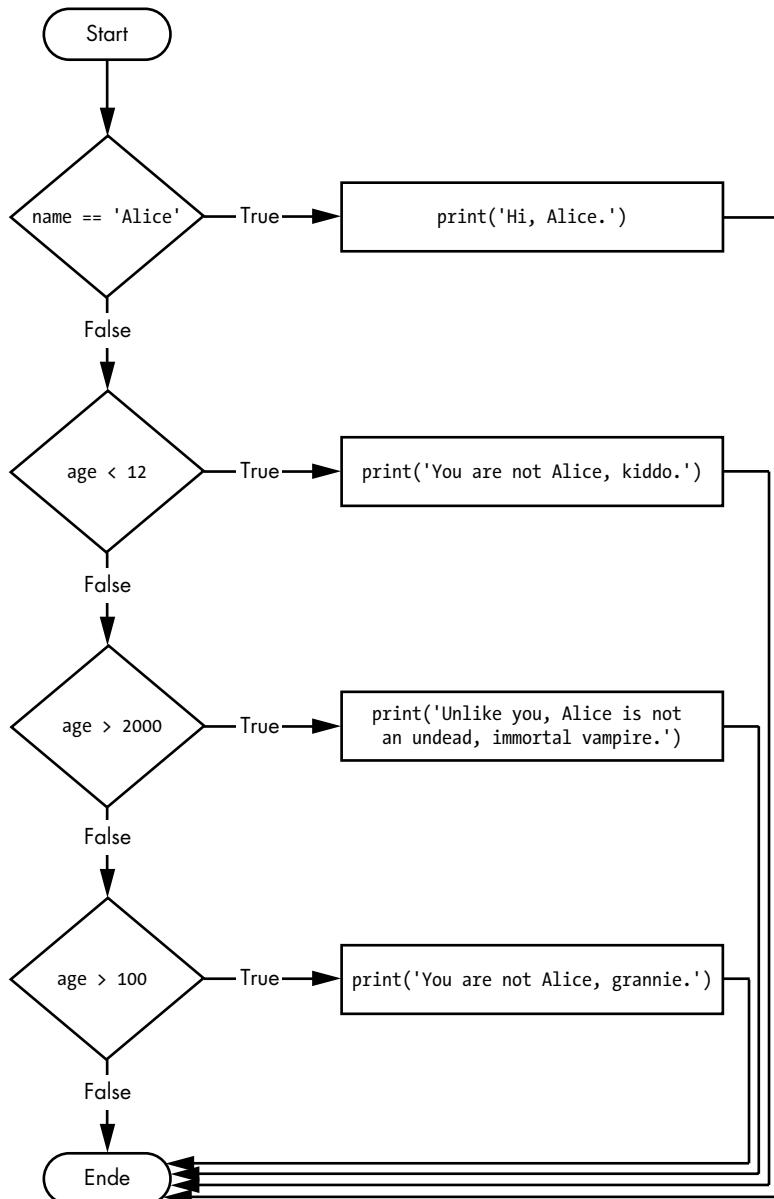
**Abb. 2–5** Flussdiagramm einer `elif`-Anweisung

Die `elif`-Klausel wird ausgeführt, wenn `age < 12` den Wert `True` ergibt und `name == 'Alice'` den Wert `False`. Wenn aber beide Bedingungen `False` sind, dann werden beide Klauseln übersprungen. Es gibt *keine* Garantie dafür, dass wenigstens eine ausgeführt wird. Bei einer Kette von `elif`-Anweisungen wird nur eine einzige von ihnen oder gar keine ausgeführt. Sobald eine Bedingung dieser Anweisungen zu `True` ausgewertet wird, werden alle restlichen `elif`-Klauseln automatisch verworfen. Um sich das in einem Beispiel anzusehen, öffnen Sie ein neues Dateieditorfenster, geben den folgenden Code ein und speichern ihn als *vampire.py*:

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.)
```

Hier habe ich zwei weitere `elif`-Anweisungen hinzugefügt, damit das Programm die Person auf der Grundlage des Werts von `age` mit verschiedenen Meldungen begrüßt. Das Flussdiagramm dafür sehen Sie in Abb. 2–6.

Dabei kommt es jedoch auf die Reihenfolge der `elif`-Anweisungen an. Um das zu zeigen, ordnen wir sie um, sodass sich ein Fehler ergibt. Da alle restlichen `elif`-Klauseln verworfen werden, wenn eine wahre Bedingung gefunden wurde, führt eine Umsortierung der Klauseln in *vampire.py* zu einem Problem. Ändern Sie den Code wie folgt und speichern Sie diese Version als *vampire2.py*:



**Abb. 2-6** Flussdiagramm für das Programm `vampire.py` mit mehreren `elif`-Anweisungen

```

if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 100: ❶
    print('You are not Alice, grannie.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')

```

Nehmen Sie nun an, die Variable `age` enthält vor der Ausführung dieses Codes den Wert 3000. Man könnte glauben, dass das Programm deshalb den String `'Unlike you, Alice is not an undead, immortal vampire.'` ausgibt. Allerdings hat schon die Bedingung `age > 100` den Wert True (schließlich ist 3000 größer als 100) (❶). Daher wird der String `'You are not Alice, grannie.'` ausgegeben und der Rest der `elif`-Anweisungen übersprungen. Merken Sie sich, dass höchstens eine Klausel ausgeführt wird und dass die Reihenfolge der `elif`-Anweisungen eine Rolle spielt.

Abb. 2–7 zeigt das Flussdiagramm für den vorstehenden Code. Die Rauten für `age > 100` und `age > 2000` sind hier vertauscht.

Optional können Sie hinter der letzten `elif`-Anweisung noch eine `else`-Anweisung einfügen. In diesem Fall wird *garantiert* mindestens eine (genau eine) Klausel ausgeführt. Wenn die Bedingungen in sämtlichen `if`- und `elif`-Anweisungen `False` sind, wird die `else`-Klausel ausgeführt. Bauen wir unsere Alice-Beispiel noch ein wenig um, sodass es `if`-, `elif`- und `else`-Klauseln verwendet:

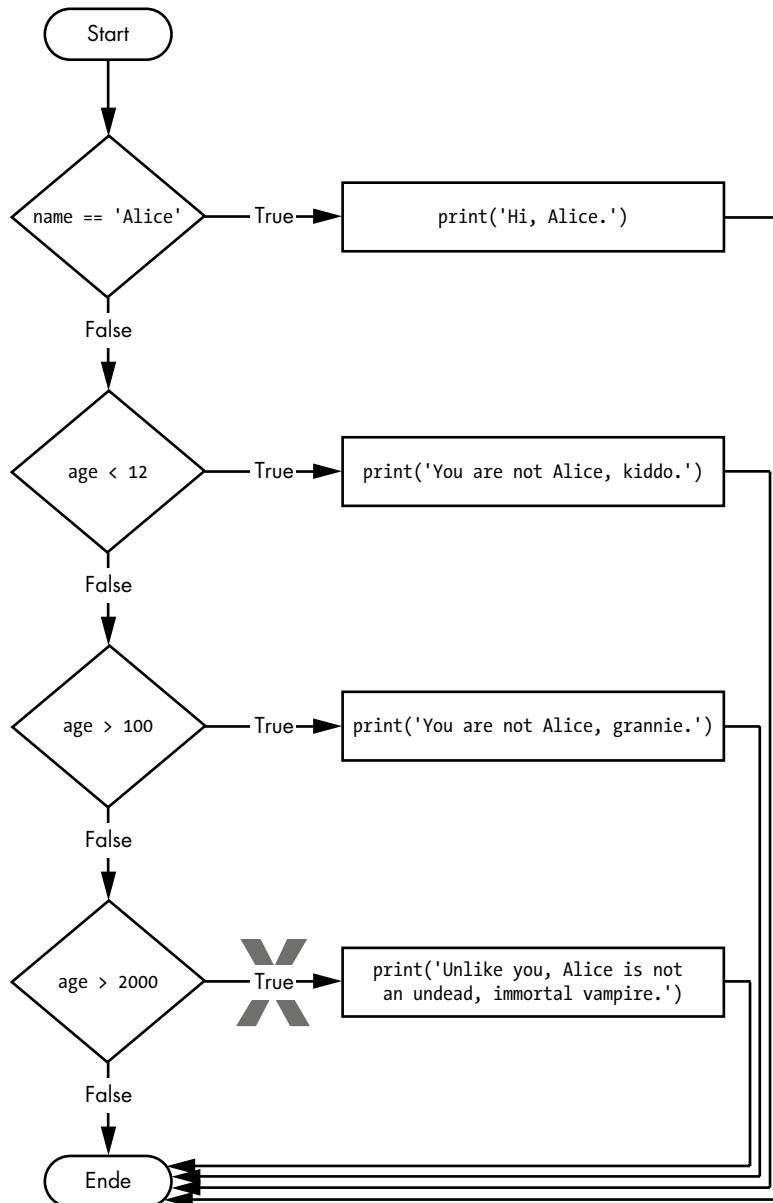
```

if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else:
    print('You are neither Alice nor a little kid.')

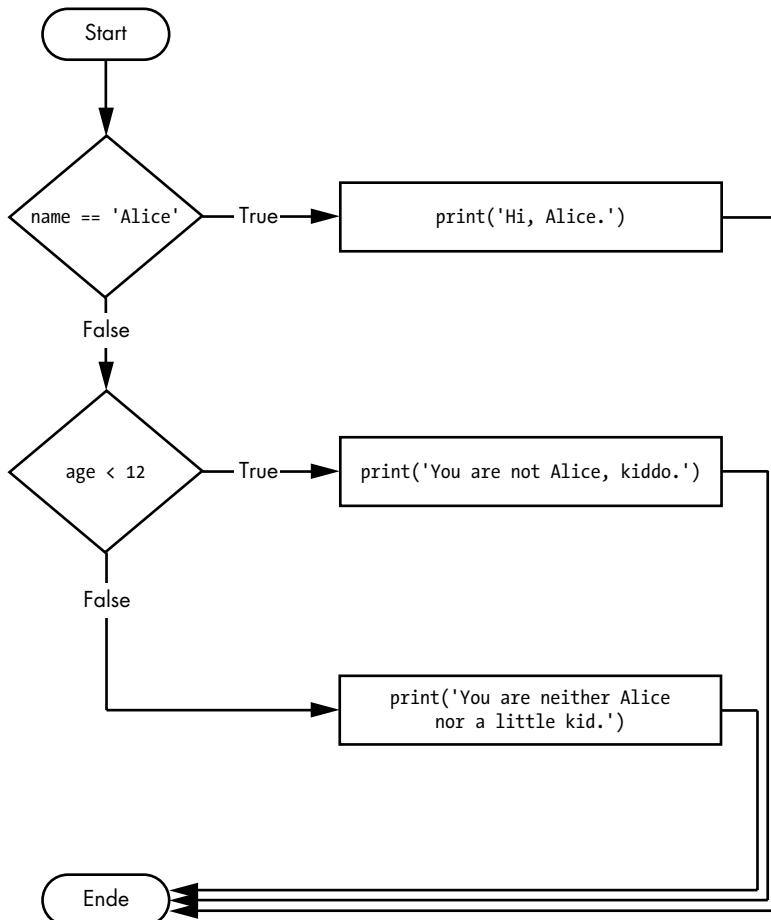
```

Diesen Code speichern wir als `littleKid.py`. Das Flussdiagramm sehen Sie in Abb. 2–8.

In Alltagssprache formuliert, lautet diese Flusssteuerungsstruktur wie folgt: »Wenn die erste Bedingung wahr ist, tu dies. Ist dagegen die zweite Bedingung wahr, tu jenes. Andernfalls mach etwas anderes.« Wenn Sie alle drei Anweisungen kombinieren, müssen Sie sich an die Regeln für die Reihenfolge halten, damit Sie keine Fehler wie den aus Abb. 2–7 einbauen. Erstens gibt es immer genau eine `if`-Anweisung. Jegliche `elif`-Anweisungen, die Sie brauchen, folgen auf die `if`-Anweisungen. Zweitens: Wenn Sie sicherstellen wollen, dass mindestens eine Klausel ausgeführt wird, müssen Sie die Struktur mit einer `else`-Anweisung abschließen.



**Abb. 2-7** Das Flussdiagramm für das Programm `vampire2.py`. Der durchgestrichene Pfad kann logisch nie erreicht werden, denn wenn der Wert von `age` größer als 2000 ist, so ist er bereits größer als 100.



**Abb. 2–8** Das Flussdiagramm für das Programm *littleKid.py*.

## While-Schleifen

Mit einer `while`-Anweisung können Sie dafür sorgen, dass der Code in der `while`-Klausel immer und immer wieder ausgeführt wird, solange die Bedingung der Anweisung `True` ist. Eine `while`-Anweisung enthält Folgendes:

- Das Schlüsselwort `while`
- Eine Bedingung (ein Ausdruck, der zu `True` oder `False` ausgewertet wird)
- Einen Doppelpunkt
- Einen eingerückten Codeblock (die `while`-Klausel), der in der nächsten Zeile beginnt

Eine `while`-Anweisung sieht ähnlich aus wie eine `if`-Anweisung, verhält sich aber ganz anders. Am Ende einer `if`-Klausel fährt das Programm mit der Ausführung hinter der `if`-Anweisung fort, wohingegen es am Ende einer `while`-Klausel zurück zum Beginn der `while`-Anweisung springt. Die `while`-Klausel wird oft auch als `while`-Schleife oder einfach als *Schleife* bezeichnet.

Zum Vergleich wollen wir uns eine `if`- und eine `while`-Anweisung ansehen, die die gleiche Bedingung verwenden und auf der Grundlage dieser Bedingung auch die gleichen Aktionen ausführen. Der Code der `if`-Anweisung sieht wie folgt aus:

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Die `while`-Anweisung stellt sich folgendermaßen dar:

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Die Anweisungen sehen ähnlich aus: Sowohl `if` als auch `while` prüfen den Wert von `spam` und geben eine Meldung aus, wenn dieser Wert kleiner als 5 ist. Aber wenn Sie diese beiden Codefragmente ausführen, geschieht jeweils etwas völlig anderes. Bei der `if`-Anweisung lautet die Ausgabe schlicht `Hello, world.`, doch bei der `while`-Anweisung wird dieser Satz fünfmal ausgegeben. Um zu sehen, warum das so ist, schauen Sie sich die Flussdiagramme für diese beiden Codefragmente in Abb. 2–9 und 2–10 an.

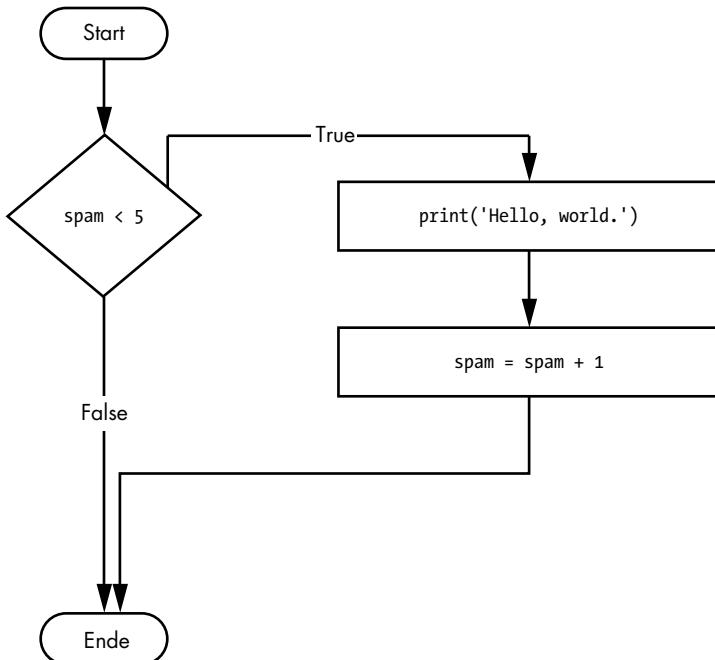


Abb. 2–9 Das Flussdiagramm für den Code mit der if-Anweisung

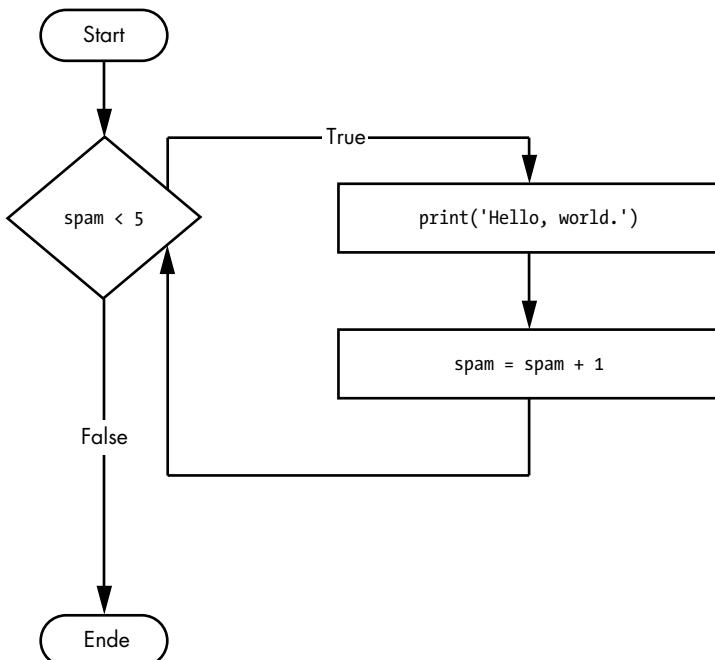


Abb. 2–10 Das Flussdiagramm für den Code mit der while-Anweisung

Der Code in der `if`-Anweisung prüft die Bedingung und gibt die Meldung `Hello, world.` ein einziges Mal aus, wenn die Bedingung wahr ist. Der Code in der `while`-Schleife dagegen gibt die Meldung fünfmal aus. Danach hält er an, da der Integerwert in `spam` nach jedem Schleifendurchlauf um 1 erhöht wird. Die Schleife wird also fünfmal ausgeführt, bevor `spam < 5` falsch ist.

In einer `while`-Schleife wird die Bedingung zu Beginn jeder *Iteration* (also vor jedem Durchlauf durch die Schleife) geprüft. Ist sie `True`, so wird die Klausel ausgeführt. Danach wird die Bedingung erneut geprüft. Wenn die Bedingung zum ersten Mal `False` ist, wird die `while`-Klausel übersprungen.

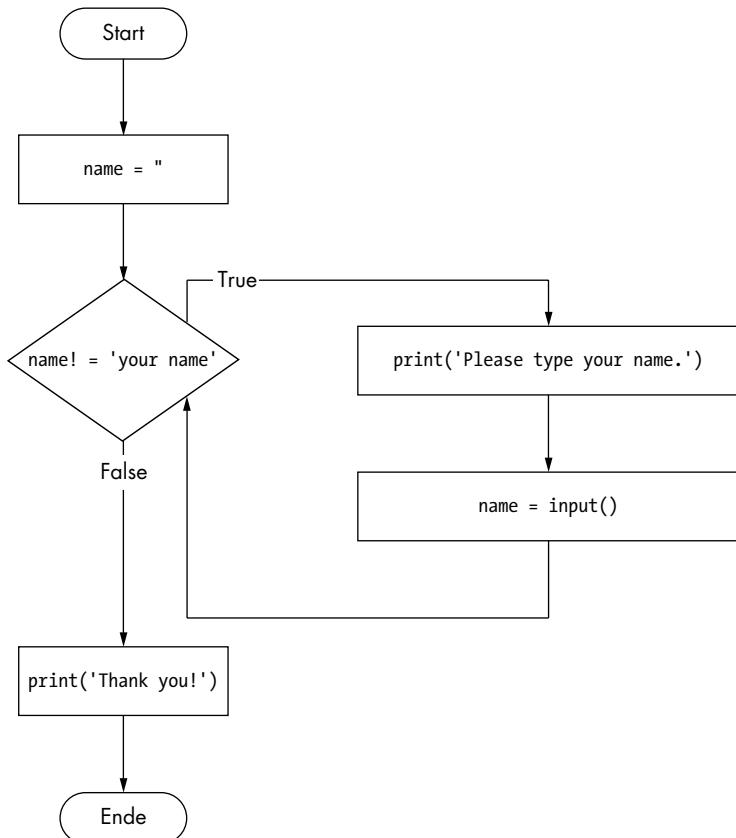
Das folgende kleine Beispielprogramm fordert Sie immer wieder auf, »`your name`« einzugeben, bis Sie tatsächlich wortwörtlich »`your name`« eingeben (und nicht etwa Ihren Namen). Öffnen Sie mit *File > New Window* ein neues Dateieditorfenster, geben Sie den folgenden Code ein und speichern Sie die Datei als `yourName.py`:

```
name = ''    ❶
while name != 'your name':    ❷
    print('Please type your name.')
    name = input()    ❸
print('Thank you!')    ❹
```

Zu Anfang setzt das Programm die Variable `name` auf einen leeren String (❶), damit die Bedingung `name != 'your name'` zu `True` ausgewertet wird und das Programm mit der Ausführung der `while`-Klausel beginnt (❷).

Der Code in dieser Klausel fordert den Benutzer auf, seinen Namen einzugeben, und diese Eingabe wird der Variablen `name` zugewiesen (❸). Da dies die letzte Zeile des Blocks ist, kehrt die Ausführung wieder zum Anfang der `while`-Schleife zurück, wo die Bedingung erneut ausgewertet wird. Wenn der Wert in `name ungleich` dem String `'your name'` ist, so ist die Bedingung `True`, weshalb die `while`-Klausel erneut ausgeführt wird.

Wenn der Benutzer aber `your name` eingibt, lautet die Bedingung `'your name' != 'your name'`, was `False` ist. Daher wird die `while`-Schleife nicht erneut ausgeführt, sondern übersprungen, sodass die Ausführung mit dem Rest des Programms fortgesetzt wird (❹). Abbildung 2–11 zeigt das Flussdiagramm des Programms `yourName.py`.



**Abb. 2-11** Das Flussdiagramm für das Programm `yourName.py`

Schauen wir uns `yourName.py` nun in Aktion an. Drücken Sie **[F5]**, um das Programm auszuführen, und geben Sie mehrere Male irgendetwas anderes als `your name` ein, bevor Sie dem Programm schließlich das geben, was es haben will.

```

Please type your name.
A1
Please type your name.
Albert
Please type your name.
%#@%*(^&!!!
Please type your name.
your name
Thank you!
  
```

Wenn Sie niemals `your name` eingeben, ist die Bedingung der `while`-Schleife niemals `False`. Das Programm wird Sie dann bis in alle Ewigkeit zur Eingabe auffordern. Der Aufruf von `input()` stellt für den Benutzer die Möglichkeit bereit, den richtigen

String einzugeben, damit das Programm aus der Schleife herauskommt. In einem Programm kann es jedoch durchaus vorkommen, dass sich die Bedingung niemals ändert, was ein ziemliches Problem darstellen kann. Sehen wir uns daher an, wie Sie aus einer `while`-Schleife ausbrechen können.

### Break-Anweisungen

Es gibt eine Abkürzung, um vorzeitig aus einer `while`-Schleife auszubrechen. Wenn das Programm bei der Ausführung eine `break`-Anweisung erreicht, verlässt es die `while`-Klausel sofort. Im Code besteht eine `break`-Anweisung lediglich aus dem Schlüsselwort `break`.

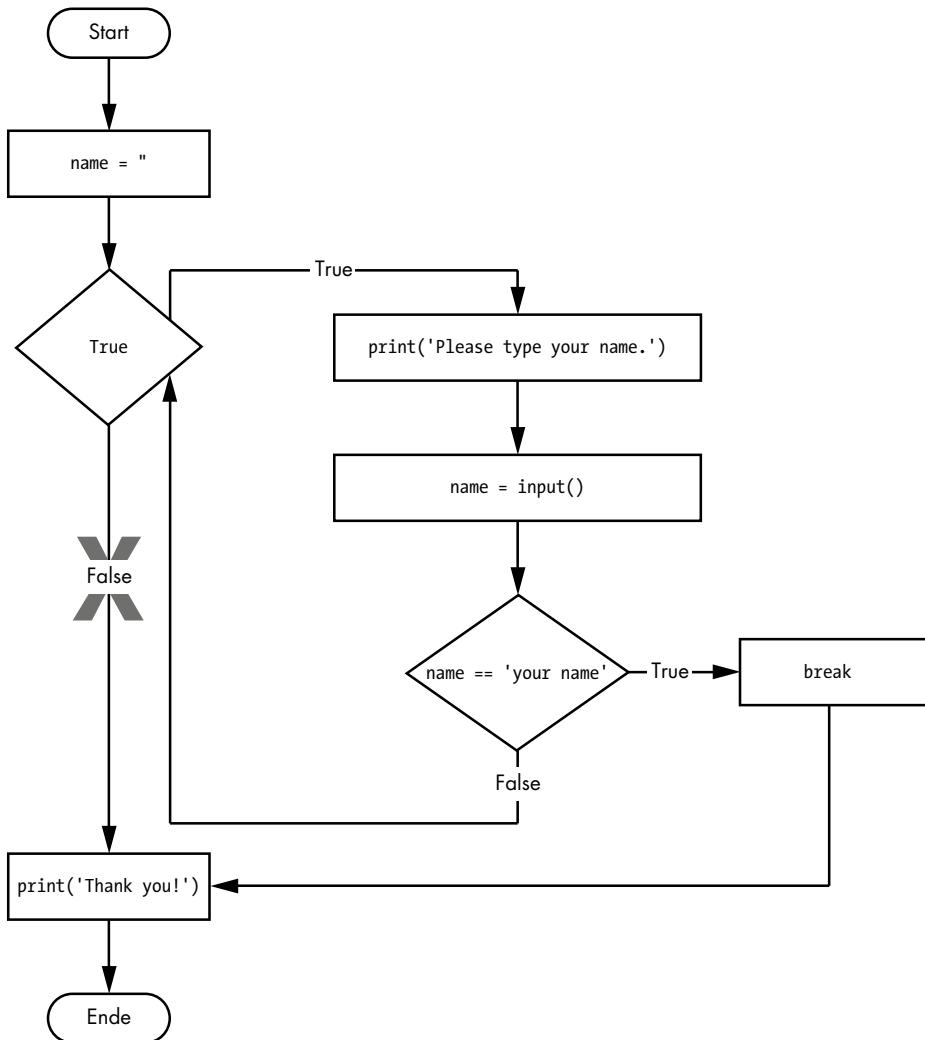
Ziemlich einfach, oder? Das folgende Programm macht das Gleiche wie das vorherige, enthält zusätzlich aber eine `break`-Anweisung, um aus der Schleife ausbrechen zu können. Geben Sie den folgenden Code ein und speichern Sie die Datei als `yourName2.py`:

```
while True:    ❶
    print('Please type your name.')
    name = input()    ❷
    if name == 'your name':    ❸
        break    ❹
    print('Thank you!')    ❺
```

Die erste Zeile (❶) ruft eine *Endlosschleife* hervor – eine `while`-Schleife, deren Bedingung stets `True` ist (denn schließlich wird der Ausdruck `True` immer zu dem Wert `True` ausgewertet). Das Programm tritt in die Schleife ein und verlässt sie erst, wenn eine `break`-Anweisung ausgeführt wird. (Eine Endlosschleife, die *niemals* verlassen wird, ist ein häufig auftretender Bug.)

Wie zuvor fordert das Programm den Benutzer dazu auf, `your name` einzugeben (❷). Diesmal aber wird innerhalb der `while`-Schleife eine `if`-Anweisung ausgeführt (❸), um zu prüfen, ob `name` gleich '`your name`' ist. Ist diese Bedingung erfüllt, so wird die `break`-Anweisung ausgeführt (❹), sodass das Programm die Schleife verlässt und zu `print('Thank you!')` übergeht (❺). Andernfalls wird die `if`-Klausel mit der `break`-Anweisung übersprungen, sodass die Programmausführung das Ende der `while`-Schleife erreicht. Dadurch springt sie wieder zum Anfang der `while`-Anweisung, wo die Bedingung überprüft wird (❶). Da es sich bei dieser Bedingung einfach um den booleschen Wert `True` handelt, beginnt die Schleife erneut, sodass der Benutzer wieder aufgefordert wird, `your name` einzugeben. Das Flussdiagramm für dieses Programm sehen Sie in Abb. 2–12.

Führen Sie `yourName2.py` aus und geben Sie den gleichen Text ein wie bei `yourName.py`. Die neue Version des Programms reagiert auf die gleiche Weise wie die ursprüngliche.



**Abb. 2–12** Das Flussdiagramm für das Programm `yourName2.py` mit einer Endlosschleife. Der mit X markierte Pfad wird nie erreicht, da die Schleifenbedingung stets True ist.

### Continue-Anweisungen

Ebenso wie `break`-Anweisungen werden auch `continue`-Anweisungen innerhalb von Schleifen verwendet. Wird bei der Ausführung eine solche Anweisung erreicht, so erfolgt ein Rücksprung zum Anfang der Schleife, wo die Schleifenbedingung erneut ausgewertet wird. (Dies geschieht auch, wenn die Ausführung am Ende der Schleife ankommt.)

### Gefangen in einer Endlosschleife?

Wenn Sie ein Programm ausführen und es aufgrund eines Bugs in einer Endlosschleife hängen bleibt, drücken Sie **Strg** + **C**. Dadurch wird ein KeyboardInterrupt an das Programm gesendet und es bricht sofort ab. Um das auszuprobieren, können Sie im Dateieditor eine Endlosschleife schreiben und als *infiniteloop.py* speichern:

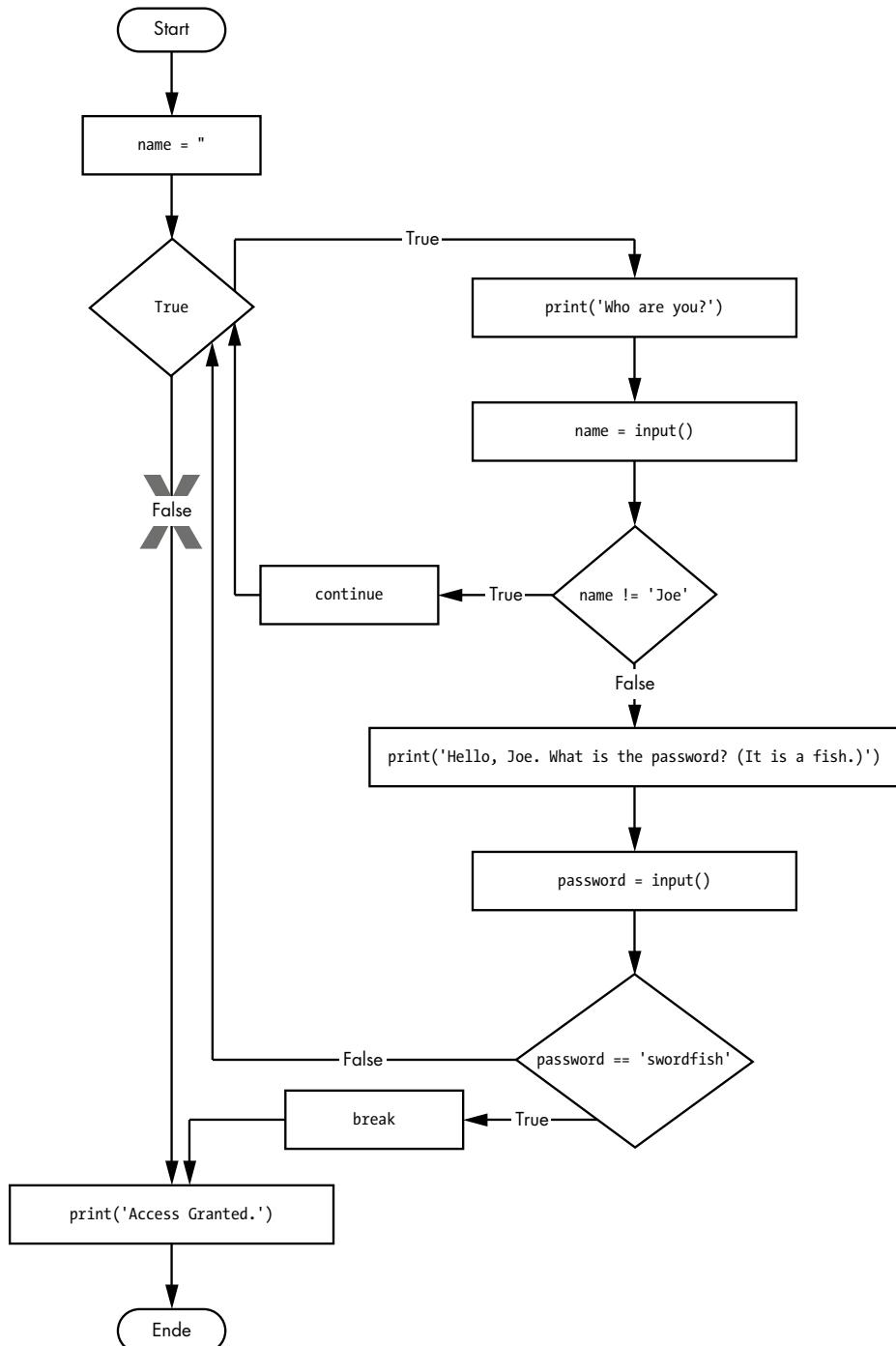
```
while True:
    print('Hello world!')
```

Wenn Sie dieses Programm ausführen, gibt es *Hello world!* auf dem Bildschirm aus und will nicht mehr damit aufhören, da die Bedingung der while-Anweisung stets True ist. In der interaktiven Shell von IDLE gibt es nur zwei Möglichkeiten, dieses Programm abzubrechen: Drücken Sie entweder **Strg** + **C** oder wählen Sie im Menü *Shell > Restart Shell*. **Strg** + **C** ist auch dann praktisch, wenn ein Programm nicht in einer Endlosschleife gefangen ist, Sie es aber aus irgendeinem anderen Grund sofort abbrechen wollen.

Im Folgenden verwenden wir *continue* für ein Programm, das Sie nach Ihrem Namen und Ihrem Passwort fragt. Geben Sie den folgenden Code in ein neues Dateieditorfenster ein und speichern Sie das Programm als *swordfish.py*:

```
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe': ❶
        continue ❷
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input() ❸
    if password == 'swordfish':
        break ❹
    print('Access granted.') ❺
```

Wenn der Benutzer irgendeinen anderen Namen als Joe eingibt (❶), sorgt die *continue*-Anweisung (❷) dafür, dass die Programmausführung zum Beginn der Schleife zurückspringt. Nach der Auswertung der Bedingung wird immer die Schleife ausgeführt, da es sich bei dieser Bedingung einfach um den Wert True handelt. Hat die Ausführung erst einmal die erste *if*-Anweisung überwunden, wird der Benutzer nach dem Passwort gefragt (❸). Gibt der Benutzer *swordfish* ein, wird die *break*-Anweisung ausgeführt (❹), sodass das Programm die *while*-Schleife verlässt und die Meldung *Access granted.* ausgibt (❺). Andernfalls wird die Ausführung bis zum Ende der *while*-Schleife fortgesetzt, von wo aus sie zum Beginn der Schleife zurückspringt. Das können Sie in dem Flussdiagramm in Abb. 2–13 genauer erkennen.



**Abb. 2-13** Das Flussdiagramm für das Programm `swordfish.py`. Der mit X markierte Pfad wird nie erreicht, da die Schleifenbedingung stets True ist.

### »Truthy«- und »Falsey«-Werte

Es gibt auch einige Werte anderer Datentypen, die in Bedingungen als gleichwertig mit True und False angesehen werden. In Bedingungen gelten 0, 0.0 und '' (ein leerer String) als False, alle anderen Werte dagegen als True. Betrachten Sie beispielsweise das folgende Programm:

```
name = ''  
while not name: ❶  
    print('Enter your name:')  
    name = input()  
print('How many guests will you have?')  
numOfGuests = int(input())  
if numOfGuests: ❷  
    print('Be sure to have enough room for all your guests.') ❸  
print('Done')
```

Wenn der Benutzer einen leeren String für name eingibt, dann ist die Bedingung der while-Anweisung True (❶), sodass das Programm weiterhin nach dem Namen fragt. Ist der Wert von numOfGuests ungleich 0 (❷), wird die Bedingung als True aufgefasst, sodass das Programm einen Hinweis für den Benutzer ausgibt (❸).

Sie könnten auch `not name != ''` statt `not name` und `numOfGuests != 0` statt `numOfGuests` schreiben, aber mit den sogenannten Truthy- und Falsey-Werten wird der Code übersichtlicher.

Führen Sie das Programm aus und machen Sie einige Eingaben. Erst wenn Sie sich als Joe vorstellen, werden Sie nach dem Passwort gefragt. Nachdem Sie das richtige Passwort eingegeben haben, endet das Programm.

```
Who are you?  
I'm fine, thanks. Who are you?  
Who are you?  
Joe  
Hello, Joe. What is the password? (It is a fish.)  
Mary  
Who are you?  
Joe  
Hello, Joe. What is the password? (It is a fish.)  
swordfish  
Access granted.
```

## For-Schleifen und die Funktion range()

While-Schleifen werden durchlaufen, solange die Bedingung True ist. Was aber machen Sie, wenn Sie einen Codeblock eine bestimmte Anzahl von Malen ausführen wollen? Das können Sie mit der Anweisung for und der Funktion range() erreichen.

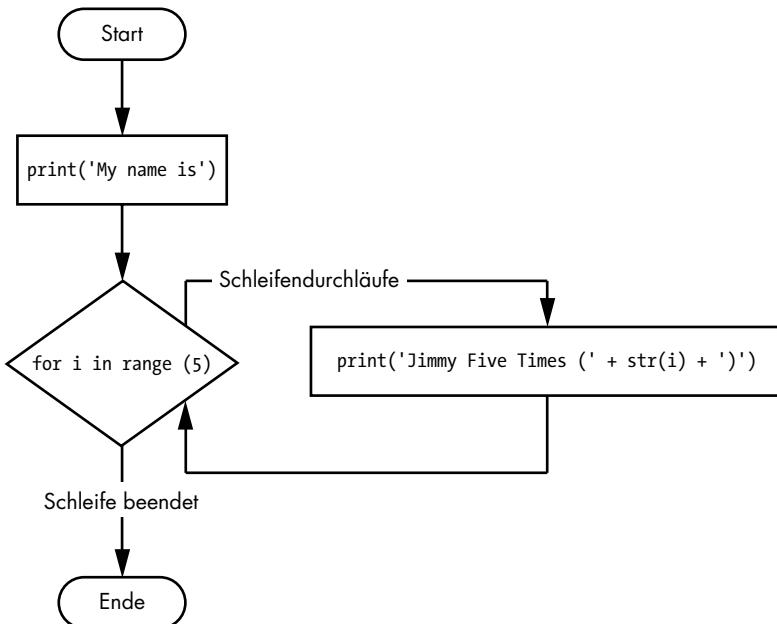
Eine for-Anweisung weist die Form for i in range(5) auf und besteht aus folgenden Elementen:

- Dem Schlüsselwort for
- Einem Variablenamen
- Dem Schlüsselwort in
- Einem Aufruf der Methode range(), wobei bis zu drei Integer übergeben werden können
- Einem Doppelpunkt
- Einem eingerückten Codeblock (die for-Klausel), der in der nächsten Zeile beginnt

Um uns eine for-Schleife in Aktion anzusehen, schreiben wir das Programm *fiveTimes.py*:

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

Der Code in der for-Klausel wird fünfmal ausgeführt. Beim ersten Mal hat die Variable i den Wert 0. Der Aufruf von print() in der Klausel gibt daher Jimmy Five Times (0) aus. Nach einem kompletten Durchlauf durch den Code in der Klausel wird die Ausführung wieder am Anfang der Schleife fortgesetzt und i um 1 erhöht. Die Funktion range(5) ruft fünf Iterationen durch die Klausel hervor, da i nacheinander auf 0, 1, 2, 3 und 4 gesetzt wird. Die Variable wird erhöht, bis die nächste Erhöhung den an range() übergebenen Wert erreichen würde. Das Flussdiagramm für *fiveTimes.py* sehen Sie in Abb. 2–14.



**Abb. 2-14** Das Flussdiagramm für `fiveTimes.py`

Wenn Sie das Programm ausführen, gibt es fünfmal Jimmy Five Times gefolgt von dem Wert von `i` aus, bevor es die Schleife verlässt.

```

My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
  
```

### Hinweis

Auch in `for`-Schleifen können Sie die Anweisungen `break` und `continue` verwenden. Dabei sorgt eine `continue`-Anweisung dafür, dass das Programm mit dem nächsten Wert des Schleifenzählers fortfährt, als wäre die Ausführung am Ende der Schleife angekommen und zum Start zurückgesprungen. `Break`- und `continue`-Anweisungen lassen sich ausschließlich in `while`- und `for`-Schleifen einsetzen. Wenn Sie versuchen, sie an anderen Stellen zu verwenden, gibt Python eine Fehlermeldung aus.

Unser nächstes Beispiel für eine `for`-Schleife geht auf eine Geschichte über den Mathematiker Carl Friedrich Gauß zurück. Als er ein kleiner Junge war, wollte sein Lehrer die Klasse beschäftigt halten und wies die Schüler daher an, alle Zah-

len von 1 bis 100 zu addieren. Der kleine Gauß kam aber auf einen Trick, mit dem er das Ergebnis in wenigen Sekunden berechnen konnte. In Python brauchen Sie diesen Trick gar nicht, sondern Sie können die Arbeit einfach in einer `for`-Schleife erledigen lassen:

```
total = 0    ①
for num in range(101):  ②
    total = total + num  ③
print(total)  ④
```

Das Ergebnis lautet 5050. Zu Anfang des Programms wird die Variable `total` auf 0 gesetzt (①). Die `for`-Schleife (②) führt dann 100 Mal `total = total + num` (③) aus. Wenn die Schleife alle 100 Iterationen durchlaufen hat, sind alle ganzen Zahlen von 1 bis 100 zu `total` addiert worden. Jetzt wird der Wert von `total` auf dem Bildschirm ausgegeben (④). Selbst auf dem langsamsten Computer nimmt dieses Programm weniger als eine Sekunde in Anspruch.

(Der kleine Gauß hatte übrigens festgestellt, dass die Zahlen von 1 bis 100 aus 50 Paaren mit der Summe 101 bestanden; 1 + 100, 2 + 99, 3 + 98 usw. bis 50 + 51. Daher musste er nur noch  $101 \times 50 = 5050$  rechnen. Schlaues Bürschchen!)

### Eine gleichwertige `while`-Schleife

Es ist übrigens möglich, eine `while`-Schleife zu konstruieren, die genau das Gleiche macht wie eine `for`-Schleife; die `for`-Schleife stellt nur eine knappere Schreibweise dar. Die folgende Variante von *fiveTimes.py* verwendet eine `while`- statt der `for`-Schleife:

```
print('My name is')
i = 0
while i < 5:
    print('Jimmy Five Times (' + str(i) + ')')
    i = i + 1
```

Wenn Sie dieses Programm ausführen, erhalten Sie die gleiche Ausgabe wie bei dem Programm *fiveTimes.py* mit der `for`-Schleife.

### Das Anfangs-, End- und Schrittargument für `range()`

Manche Funktionen können mit mehreren, durch Kommata getrennten Argumenten aufgerufen werden. Unter anderem ist das auch bei `range()` möglich. Damit können Sie dafür sorgen, dass die Funktion beliebige Intervalle durchläuft, auch solche, die nicht mit 0 beginnen:

```
for i in range(12, 16):
    print(i)
```

Das erste Argument gibt den Wert an, mit dem die Variable der `for`-Schleife beginnt, das zweite die Zahl, bis zu der die Inkrementierung durchgeführt wird (ohne sie zu erreichen).

```
12
13
14
15
```

Die Funktion `range()` kann auch mit drei Argumenten aufgerufen werden. Dabei sind die ersten beiden wie gehabt der Anfangs- und der Endwert, während der dritte die *Schrittweite* angibt, also den Wert, um den die Variable nach jeder Iteration erhöht wird.

```
for i in range(0, 10, 2):
    print(i)
```

Wenn Sie also `range(0, 10, 2)` aufrufen, wird in Abständen von 2 von 0 bis 8 gezählt:

```
0
2
4
6
8
```

Was die Reihenfolge der Zahlen für die `for`-Schleifen angeht, ist die Funktion `range()` sehr flexibel. Sie können auch eine negative Zahl als Schrittweite angeben, damit die `for`-Schleife abwärts zählt statt aufwärts.

```
for i in range(5, -1, -1):
    print(i)
```

Wenn Sie in einer `for`-Schleife den Wert von `i` ausgeben und dabei `range(5, -1, -1)` angeben, wird von 5 bis 0 abwärts gezählt:

```
5
4
3
2
1
0
```

## Module importieren

Alle Python-Programmen können einen grundlegenden Satz von eingebauten oder *integrierten Funktionen* nutzen, beispielsweise die Funktionen `print()`, `input()` und `len()`, die Sie schon kennengelernt haben. Darüber hinaus wird Python mit einem Satz von Modulen geliefert, die als *Standardbibliothek* bezeichnet werden. Jedes Modul ist ein Python-Programm mit einer Gruppe verwandter Funktionen, die Sie in Ihr Programm einbetten können. Beispielsweise verfügt das Modul `math` über mathematische Funktionen, das Modul `random` über Funktionen im Zusammenhang mit Zufallszahlen usw.

Bevor Sie die Funktionen in einem Modul nutzen können, müssen Sie das Modul mit der Anweisung `import` importieren. Diese Anweisung enthält folgende Elemente:

- Das Schlüsselwort `import`
- Der Name des Moduls
- Optional weitere Modulnamen, die durch Kommata getrennt sind

Nachdem Sie ein Modul importiert haben, können Sie alle seine Funktionen nutzen. Das wollen wir anhand des Moduls `random` ausprobieren, das uns Zugriff auf die Funktion `random.randint()` gibt.

Geben Sie den folgenden Code in den Dateieditor ein und speichern Sie ihn als `printRandom.py`:

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

Wenn Sie dieses Programm ausführen, erhalten Sie eine Ausgabe wie die folgende:

```
4
1
8
4
1
```

Der Aufruf der Funktion `random.randint()` wird zu einem zufälligen Integerwert ausgewertet, der zwischen den beiden übergebenen Integern liegt. Da `randint()` zum Modul `random` gehört, müssen Sie dem Funktionsnamen `random.` voranstellen, um Python anzuweisen, in diesem Modul nach der Funktion zu suchen.

Das folgende Beispiel zeigt eine `import`-Anweisung, die gleich vier verschiedene Module importiert:

```
import random, sys, os, math
```

Danach können Sie alle Funktionen aus diesen vier Modulen nutzen. Mehr darüber erfahren Sie weiter hinten in diesem Buch.

### From-import-Anweisungen

Eine alternative Form der `import`-Anweisung besteht aus dem Schlüsselwort `from` gefolgt vom Modulnamen, dem Schlüsselwort `import` und einem Sternchen, also z. B. `from random import *`.

Bei dieser Form müssen Sie Aufrufen der Funktionen aus `random` das Präfix `random.` nicht mehr voranstellen. Allerdings ist der Code besser verständlich, wenn Sie den vollständigen Namen verwenden, weshalb es besser ist, die übliche Form der `import`-Anweisung zu nehmen.

### Programme mit `sys.exit()` vorzeitig beenden

Der letzte Aspekt der Flusssteuerung, mit dem wir uns hier beschäftigen, ist die Beendigung eines Programms. Wenn die Ausführung am Ende der Anweisungen angelangt ist, wird das Programm automatisch beendet, allerdings können Sie durch einen Aufruf der Funktion `sys.exit()` auch ausdrücklich dafür sorgen, dass das Programm beendet wird. Da diese Funktion zum Modul `sys` gehört, müssen Sie dieses zunächst in Ihr Programm importieren, bevor Sie die Funktion nutzen können.

Geben Sie in einem neuen Dateieditorfenster den folgenden Code ein und speichern Sie ihn als `exitExample.py`:

```
import sys

while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

Führen Sie das Programm in IDLE aus. Es enthält eine Endlosschleife ohne `break`-Anweisung. Es wird nur beendet, wenn `response` gleich `exit` ist, denn dadurch wird `sys.exit()` aufgerufen. Da die Variable `response` aber durch die Funktion `input()` festgelegt wird, muss der Benutzer also `exit` eingeben, um das Programm zu beenden.

## **Zusammenfassung**

Mithilfe von Ausdrücken, die zu True oder False ausgewertet werden (sogenannten *Bedingungen*), können Sie Programme schreiben, die entscheiden, welcher Code ausgeführt und welcher übersprungen werden soll. Es ist auch möglich, Code in einer Schleife wiederholt auszuführen, während eine bestimmte Bedingung True ist. Wenn Sie eine Schleife abbrechen oder innerhalb einer Schleife zum Anfang zurückkehren müssen, können Sie auf die Anweisung break bzw. continue zurückgreifen.

Diese Flusssteuerungsanweisungen erlauben es Ihnen, viel intelligenterer Programme zu schreiben. Es gibt jedoch noch eine andere Art von Flusssteuerung, die Sie dadurch erreichen, dass Sie eigene Funktionen schreiben. Das ist das Thema des nächsten Kapitels.

## **Wiederholungsfragen**

1. Welche beiden Werten gibt es für den booleschen Datentyp? Wie werden sie geschrieben?
2. Wie heißen die drei booleschen Operatoren?
3. Schreiben Sie die Wahrheitswertetafeln aller booleschen Operatoren auf (also die Tabellen aller möglichen Kombinationen der booleschen Werte für diese Operatoren und das jeweilige Ergebnis).
4. Wie lauten die Ergebnisse der folgenden Ausdrücke?

```
(5 > 4) and (3 == 5)  
not (5 > 4)  
(5 > 4) or (3 == 5)  
not ((5 > 4) or (3 == 5))  
(True and True) and (True == False)  
(not False) or (not True)
```

5. Wie heißen die sechs Vergleichsoperatoren?
6. Was ist der Unterschied zwischen dem Gleichheits- und dem Zuweisungsoperator?
7. Was ist eine Bedingung? Wann wird sie verwendet?
8. Bezeichnen Sie die drei Blöcke in dem folgenden Code:

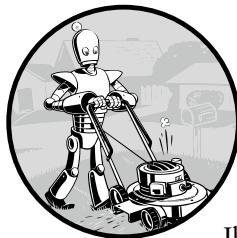
```
spam = 0
if spam == 10:
    print('eggs')
    if spam > 5:
        print('bacon')
    else:
        print('ham')
    print('spam')
print('spam')
```

9. Schreiben Sie Code, der Hello ausgibt, wenn in spam der Wert 1 gespeichert ist, Howdy, wenn spam den Wert 2 hat, und Greetings! bei jedem anderen Wert von spam.
10. Welche Tastenkombination können Sie drücken, wenn ein Programm in einer Endlosschleife gefangen ist?
11. Was ist der Unterschied zwischen break und continue?
12. Was ist der Unterschied zwischen range(10), range(0, 10) und range(0, 10, 1) in einer for-Schleife?
13. Schreiben Sie ein kurzes Programm, das die Zahlen von 1 bis 10 mithilfe einer for-Schleife ausgibt. Schreiben Sie dann ein Programm, das die gleiche Aufgabe mithilfe einer while-Schleife erfüllt.
14. Nehmen Sie an, Sie haben ein Modul namens spam, in dem sich die Funktion bacon() befindet. Wie rufen Sie bacon() auf, nachdem Sie spam importiert haben?

**Zusatzpunkt:** Schlagen Sie die Funktionen round() und abs() im Internet nach und finden Sie heraus, was sie tun. Experimentieren Sie in der interaktiven Shell damit.

# 3

## Funktionen



In den vorhergehenden Kapiteln haben Sie bereits die Funktionen `print()`, `input()` und `len()` kennengelernt. Python schließt noch weitere integrierte Funktionen wie diese ein, aber Sie können auch Ihre eigenen schreiben. Eine *Funktion* ist ein Miniprogramm innerhalb eines Programms.

Um besser zu verstehen, wie Funktionen aufgebaut sind und was sie tun, erstellen wir eine. Geben Sie das folgende Programm im Dateieditor ein und speichern Sie es als `helloFunc.py`:

```
def hello():    ❶
    print('Howdy!')    ❷
    print('Howdy!!!!')
    print('Hello there.')  
  
hello()    ❸
hello()
hello()
```

Die erste Zeile enthält die Anweisung `def` (❶), die eine Funktion namens `hello()` definiert. Der Code in dem darauf folgenden Block (❷) stellt den Rumpf der Funktion dar. Während der Funktionsdefinition wird er nicht ausgeführt, sondern erst, wenn die Funktion aufgerufen wird.

Die `hello()`-Zeilen im Anschluss an die Funktionsdefinition (❸) sind Funktionsaufrufe. Im Code wird ein solcher Aufruf in Form des Funktionsnamens gefolgt von den Klammern geschrieben, wobei in den Klammern Zahlen als Argumente stehen können. Wenn die Programmausführung diese Aufrufe erreicht, springt sie zur ersten Zeile in der Funktion und führt den dort vorhandenen Code aus. Am Ende der Funktion angelangt, kehrt die Ausführung wieder zu der Zeile zurück, in der die Funktion aufgerufen wurde. Der Code wird dann wie zuvor weiter abgearbeitet.

Da das Programm die Funktion `hello()` dreimal aufruft, wird der Code von `hello()` dreimal ausgeführt. Wenn Sie das Programm starten, sehen Sie folgende Ausgabe:

```
Howdy!  
Howdy!!!  
Hello there.  
Howdy!  
Howdy!!!  
Hello there.  
Howdy!  
Howdy!!!  
Hello there.
```

Ein wichtiger Zweck von Funktionen besteht darin, Code zusammenzufassen, der mehrfach ausgeführt wird. Ohne Funktionen müssten Sie den Code an jeder Stelle komplett einfügen, an der er benötigt wird. Das Programm sähe dann wie folgt aus:

```
print('Howdy!')  
print('Howdy!!!')  
print('Hello there.')  
print('Howdy!')  
print('Howdy!!!')  
print('Hello there.')  
print('Howdy!')  
print('Howdy!!!')  
print('Hello there.')
```

Eine solche Duplikierung von Code sollen Sie in jedem Fall vermeiden, denn wenn Sie Ihren Code irgendwann ändern – zum Beispiel, um einen Fehler zu korrigieren – müssten Sie ihn sonst an jeder einzelnen Stelle anpassen, an der Sie ihn eingefügt haben.

Je mehr Programmiererfahrung Sie haben, umso häufiger werden Sie sich daran machen, duplizierten oder kopierten Code zu entfernen. Dadurch werden Ihre Programme kürzer, leichter zu lesen und leichter zu ändern.

## Def-Anweisungen mit Parametern

Beim Aufruf von Funktionen wie `print()` und `len()` übergeben Sie Werte, sogenannte *Argumente*, indem Sie sie in die Klammern schreiben. Auch Ihre eigenen Funktionen können Sie so definieren, dass sie Argumente annehmen. Geben Sie das folgende Beispiel im Dateieditor ein und speichern Sie es als `helloFunc2.py`:

```
def hello(name):    ❶
    print('Hello ' + name) ❷

hello('Alice') ❸
hello('Bob')
```

Wenn Sie dieses Programm ausführen, erhalten Sie folgende Ausgabe:

```
Hello Alice
Hello Bob
```

Die Definition der Funktion `hello()` in diesem Programm schließt den Parameter `name` ein (❶). Ein *Parameter* ist eine Variable, in der ein Argument bei einem Funktionsaufruf gespeichert wird. Beim ersten Mal wird die Funktion `hello()` mit dem Argument `'Alice'` aufgerufen (❸). Die Programmausführung fährt mit der Funktion fort, wobei die Variable `name` automatisch auf `'Alice'` gesetzt wird. Dieser Text wird in die Ausgabe der Anweisung `print()` aufgenommen (❷).

Wenn die Funktion die Steuerung zurückgibt (wenn die Programmausführung also die Funktion verlässt und normal fortfährt), wird der in dem Parameter gespeicherte Wert jedoch vergessen. Wenn Sie in dem vorstehenden Programm hinter `hello('Bob')` die Anweisung `print(name)` einfügen, erhalten Sie die Fehlermeldung `NameError`, da es zu diesem Zeitpunkt keine Variable namens `name` mehr gibt – sie ist nach dem Abschluss des Funktionsaufrufs `hello('Bob')` zerstört worden.

Das ähnelt dem Prinzip, dass auch die Variablen im Programm nach Beendigung des Programms vergessen werden. Weiter hinten in diesem Kapitel werde ich im Zusammenhang mit dem lokalen Gültigkeitsbereich von Funktionen noch ausführlicher darauf eingehen.

## Rückgabewerte und die Anweisung return

Wenn Sie die Funktion `len()` aufrufen und ihr ein Argument wie 'Hello' übergeben, wird der Funktionsaufruf zur Länge des übergebenen Strings ausgewertet, hier also zu dem Integerwert 5. Der Wert, zu dem ein Funktionsaufruf ausgewertet wird, ist der sogenannte **Rückgabewert** der Funktion.

Wenn Sie mit `def` eine eigene Funktion erstellen, können Sie mithilfe der Anweisung `return` festlegen, was der Rückgabewert sein soll. Eine `return`-Anweisung enthält folgende Elemente:

- Das Schlüsselwort `return`
- Der Wert oder der Ausdruck, den die Funktion zurückgeben soll

Wenn Sie in der `return`-Anweisung einen Ausdruck angeben, dann ist der Rückgabewert der Wert, zu dem dieser Ausdruck ausgewertet wird. Betrachten Sie als Beispiel das folgende Programm, das, je nachdem, welche Zahl als Argument übergeben wird, einen anderen String zurückgibt. Geben Sie den folgenden Code in den Dateieditor ein und speichern Sie ihn als *magic8Ball.py*:

```
import random ①

def getAnswer(answerNumber): ②
    if answerNumber == 1: ③
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9) ④
fortune = getAnswer(r) ⑤
print(fortune) ⑥
```

Zu Anfang des Programms importiert Python das Modul random (❶). Anschließend wird die Funktion getAnswer() definiert (❷). Da dies nur die Definition der Funktion ist, aber kein Aufruf, wird der darin enthaltene Code übersprungen. Die Ausführung fährt mit dem Aufruf der Funktion random.randint() mit den beiden Argumenten 1 und 9 fort (❸). Das Ergebnis ist ein Zufallsinteger zwischen 1 und 9 (einschließlich 1 und 9) und wird in der Variablen r gespeichert.

Als Nächstes wird die Funktion getAnswer() mit r als Argument aufgerufen (❹). Die Programmausführung springt zum Anfang dieser Funktion (❺), wo der Wert r im Parameter answerNumber gespeichert wird. Abhängig von dem Wert dieses Parameters gibt die Funktion nun einen der vielen möglichen Stringwerte zurück. Die Programmausführung kehrt anschließend zu der Zeile im Programm zurück, in der getAnswer() aufgerufen wurde (❻). Der zurückgegebene String wird der Variablen fortune zugewiesen, die an den Aufruf der Funktion print() übergeben (❼) und damit auf dem Bildschirm ausgegeben wird.

Da Sie Rückgabewerte als Argumente an andere Funktionsaufrufe übergeben können, lassen sich die drei folgenden Zeilen auch abkürzen:

```
r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

Die folgende einzelne Zeile macht genau das Gleiche:

```
print(getAnswer(random.randint(1, 9)))
```

Wie Sie wissen, bestehen Ausdrücke aus Werten und Operatoren. Da ein Funktionsaufruf zu seinem Rückgabewert ausgewertet wird, kann er daher auch in einem Ausdruck verwendet werden.

## Der Wert None

In Python gibt es den Wert `None`, der die Abwesenheit eines Werts bedeutet. Dies ist der einzige Wert des Datentyps `NoneType`. (In anderen Programmiersprachen wird er auch als `null`, `nil` oder `undefined` bezeichnet.) Ebenso wie die booleschen Werte `True` und `False` muss auch `None` mit großem Anfangsbuchstaben geschrieben werden.

Dieser Nicht-Wert ist praktisch, wenn Sie etwas speichern müssen, das nicht mit einem echten Wert in einer Variablen verwechselt werden darf. Eine mögliche Anwendung für `None` ist der Rückgabewert von `print()`. Diese Funktion zeigt Text auf dem Bildschirm an, muss anderes als `len()` oder `input()` aber eigentlich nichts

zurückgeben. Da alle Funktionsaufrufe aber zu einem Rückgabewert ausgewertet werden, gibt `print()` pro forma `None` zurück. Um sich das anzusehen, geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = print('Hello!')
Hello!
>>> None == spam
True
```

Wenn eine Funktionsdefinition keine `return`-Anweisung hat, hängt Python stillschweigend `return None` an, ebenso wie eine `while`- oder `for`-Schleife implizit mit einer `continue`-Anweisung endet. Wenn Sie eine `return`-Anweisung ohne Wert geben (also nur das Schlüsselwort `return` verwenden), wird ebenfalls `None` zurückgegeben.

## Schlüsselwortargumente und `print()`

Die meisten Argumente werden anhand ihrer Position im Funktionsaufruf identifiziert. Beispielsweise ist `random.randint(1, 10)` etwas anderes als `random.randint(10, 1)`. Die erste Funktion gibt eine Zufallszahl zwischen 1 und 10 zurück, da das erste Argument die Untergrenze und das zweite die Obergrenze des Intervalls darstellt. (Dagegen führt `random.randint(10, 1)` zu einem Fehler.)

*Schlüsselwortargumente* werden dagegen durch das Schlüsselwort bezeichnet, das ihnen in dem Funktionsaufruf vorangestellt wird. Diese Art von Argumenten wird häufig für optionale Parameter verwendet. Beispielsweise können Sie bei der Funktion `print()` mit den optionalen Parametern `end` und `sep` angeben, was am Ende der Argumente ausgegeben werden soll und was dazwischen (als Trennzeichen).

Betrachten Sie das folgende Beispielprogramm:

```
print('Hello')
print('World')
```

Die Ausgabe sieht wie folgt aus:

```
Hello
World
```

Die beiden Strings werden auf getrennten Zeilen ausgegeben, da `print()` am Ende des übergebenen Strings automatisch einen Zeilenumbruch einfügt. Mit dem Schlüsselwortargument `end` können Sie dieses Verhalten jedoch ändern. Nehmen wir an, wir haben folgendes Programm:

```
print('Hello', end='')
print('World')
```

Hier sieht die Ausgabe wie folgt aus:

```
HelloWorld
```

Die Ausgabe erscheint auf einer einzigen Zeile, da hinter 'Hello' kein Zeilenumbruch mehr ausgegeben wird, sondern ein leerer String. Das ist nützlich, wenn Sie den automatischen Zeilenumbruch hinter den Funktionsaufrufen von `print()` aufheben wollen.

Wenn Sie mehrere Stringwerte an `print()` übergeben, trennt die Funktion sie automatisch durch ein Leerzeichen. Geben Sie beispielsweise Folgendes in die interaktive Shell ein:

```
>>> print('cats', 'dogs', 'mice')
cats dogs mice
```

Das Standardtrennzeichen können Sie mit dem Schlüsselwortargument `sep` ändern. Probieren Sie in der interaktiven Shell Folgendes aus:

```
>>> print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

Auch zu Ihren selbst geschriebenen Funktionen können Sie Schlüsselwortargumente hinzufügen. Dazu müssen Sie sich jedoch mit den Datentypen für Listen und Wörterbücher (Dictionarys) auskennen, die wir in den nächsten beiden Kapiteln behandeln werden. Merken Sie sich zunächst nur, dass einige Funktionen auch über optionale Schlüsselwortargumente verfügen, die Sie beim Aufruf der Funktion angeben können.

## Lokaler und globaler Gültigkeitsbereich

Parameter und Variablen, die innerhalb einer aufgerufenen Funktion zugewiesen werden, befinden sich im *lokalen Gültigkeitsbereich* der Funktion. Dagegen haben Variablen, die außerhalb aller Funktionen zugewiesen werden, einen *globalen Gültigkeitsbereich*. Variablen in einem lokalen Gültigkeitsbereich werden als *lokale Variablen* bezeichnet, Variablen im globalen Gültigkeitsbereich als *globale Variablen*. Eine Variable ist entweder lokal oder global, aber niemals beides.

Den Gültigkeitsbereich können Sie sich wie einen Behälter für Variablen vorstellen. Wird ein Gültigkeitsbereich zerstört, so werden alle Werte der darin enthaltenen Variablen vergessen. Es gibt nur einen globalen Gültigkeitsbereich. Er wird erstellt, wenn das Programm beginnt, und am Ende des Programms zerstört. Damit sind alle seine Variablen vergessen. Wäre das nicht der Fall, so würden beim nächsten Start des Programms alle Variablen immer noch die Werte aufweisen, die sie bei der letzten Ausführung hatten.

Ein lokaler Gültigkeitsbereich entsteht beim Aufruf einer Funktion. Jegliche Variablen, die in dieser Funktion zugewiesen werden, befinden sich in diesem lokalen Gültigkeitsbereich. Wenn die Funktion die Steuerung zurückgibt, wird der lokale Gültigkeitsbereich zerstört und seine Variablen werden vergessen. Beim nächsten Aufruf der Funktion sind die Werte, die beim letzten Aufruf in den lokalen Variablen gespeichert waren, nicht mehr vorhanden.

Das Prinzip der Gültigkeitsbereiche hat einige wichtige Auswirkungen:

- Code im globalen Gültigkeitsbereich kann keine lokalen Variablen nutzen.
- Ein lokaler Gültigkeitsbereich kann dagegen auf globale Variablen zugreifen.
- Code im lokalen Gültigkeitsbereich einer Funktion kann keine Variablen aus anderen Gültigkeitsbereichen nutzen.
- Sie können für zwei Variablen den gleichen Namen wählen, sofern sie sich in unterschiedlichen Gültigkeitsbereichen befinden. Es kann also beispielsweise sowohl eine lokale als auch eine globale Variable namens `spam` geben.

Warum gibt es in Python verschiedene Gültigkeitsbereiche, anstatt alle Variablen global zu machen? Wenn eine Variable durch den Code in einem bestimmten Funktionsaufruf bearbeitet wird, dann interagiert die Funktion mit dem Rest des Programms nur durch ihre Parameter und den Rückgabewert. Bei einem Fehler schränkt das die Menge der Codezeilen ein, die dafür verantwortlich sein können. Wenn in einem Programm, das ausschließlich globale Variablen enthält, ein Fehler dafür sorgt, dass einer Variablen ein falscher Wert zugewiesen wird, ist es ziemlich schwer, die entsprechende Stelle zu finden. Dieser Wert könnte überall in dem Programm zugewiesen worden sein – und das kann irgendwo in Hunderten oder gar Tausenden von Zeilen sein! Wenn aber eine lokale Variable einen falschen Wert hat, dann wissen Sie, dass er nur im Code der entsprechenden Funktion zugewiesen worden sein kann.

In kurzen Programmen ist die Verwendung globaler Variablen kein Problem, doch bei längeren Programmen sollten Sie sich lieber nicht darauf stützen.

### **Lokale Variablen können im globalen Gültigkeitsbereich nicht verwendet werden**

Wenn Sie das folgende Programm ausführen, erhalten Sie eine Fehlermeldung:

```
def spam():
    eggs = 31337
spam()
print(eggs)
```

Die Ausgabe lautet wie folgt:

```
Traceback (most recent call last):
  File "C:/test3784.py", line 4, in <module>
    print(eggs)
NameError: name 'eggs' is not defined
```

Das liegt daran, dass die Variable eggs nur in dem lokalen Gültigkeitsbereich existiert, der beim Aufruf von spam() erstellt wird. Sobald die Programmausführung spam verlässt, wird dieser lokale Gültigkeitsbereich aber zerstört, weshalb es keine Variable namens eggs mehr gibt. Wenn das Programm versucht, print(eggs) auszuführen, meldet Python, dass eggs nicht definiert ist. Wenn Sie ein wenig darüber nachdenken, ist das auch tatsächlich sinnvoll. Während sich die Programmausführung im globalen Gültigkeitsbereich bewegt, gibt es keine lokalen Gültigkeitsbereiche und damit kann es auch keine lokalen Variablen geben. Aus diesem Grunde lassen sich im globalen Gültigkeitsbereich ausschließlich globale Variablen verwenden.

### Lokale Gültigkeitsbereiche können keine Variablen aus anderen lokalen Gültigkeitsbereichen verwenden

Beim Aufruf einer Funktion wird ein neuer lokaler Gültigkeitsbereich erstellt. Das gilt auch dann, wenn die Funktion aus einer anderen Funktion heraus aufgerufen wird. Betrachten Sie dazu das folgende Programm:

```
def spam():
    eggs = 99      ❶
    bacon()       ❷
    print(eggs)   ❸

def bacon():
    ham = 101
    eggs = 0      ❹

spam()          ❺
```

Zu Beginn des Programms wird die Funktion spam() aufgerufen (❺) und damit ein lokaler Gültigkeitsbereich erstellt. Die lokale Variable eggs (❶) wird auf 99 gesetzt. Daraufhin wird die Funktion bacon() aufgerufen (❷) und ein zweiter lokaler Gültigkeitsbereich erstellt. Mehrere lokale Gültigkeitsbereiche können zur gleichen Zeit existieren. In diesem neuen lokalen Gültigkeitsbereich wird die lokale Variable ham auf 101 gesetzt. Außerdem wird eine lokale Variable namens eggs erstellt und auf 0 gesetzt (❹). Allerdings ist dies eine andere Variable als die im lokalen Gültigkeitsbereich von spam().

Wenn `bacon()` die Steuerung zurückgibt, wird der lokale Gültigkeitsbereich für diesen Aufruf zerstört. Die Programmausführung fährt mit der Funktion `spam()` fort, um den Wert von `eggs` auszugeben (❸). Da der lokale Gültigkeitsbereich für den Aufruf von `spam()` immer noch existiert, ist die Variable `eggs` nach wie vor auf 99 gesetzt. Das ist der Wert, den das Programm ausgibt.

Was lernen wir daraus? Lokale Variablen in einer Funktion sind komplett von den lokalen Variablen in einer anderen Funktion getrennt.

### Globale Variablen können von einem lokalen Gültigkeitsbereich aus gelesen werden

Betrachten Sie das folgende Programm:

```
def spam():
    print(eggs)
eggs = 42
spam()
print(eggs)
```

Da es in der Funktion `spam()` keinen Parameter namens `eggs` und auch keinen Code gibt, der `eggs` bei der Verwendung in `spam()` einen Wert zuweist, geht Python davon aus, dass hier auf die globale Variable `eggs` verwiesen wird. Daher gibt das vorstehende Programm den Wert 42 aus.

### Lokale und globale Variablen mit demselben Namen

Um sich selbst das Leben zu erleichtern, sollten Sie es tunlichst vermeiden, lokalen Variablen den gleichen Namen zu geben wie globalen Variablen oder lokalen Variablen aus einem anderen Gültigkeitsbereich. Technisch ist das in Python jedoch möglich. Um zu sehen, was in einem solchen Fall geschieht, geben Sie folgenden Code in den Dateieditor ein und speichern ihn als `sameName.py`:

```
def spam():
    eggs = 'spam local' ❶
    print(eggs) # gibt 'spam local' aus

def bacon():
    eggs = 'bacon local' ❷
    print(eggs) # gibt 'bacon local' aus
    spam()
    print(eggs) # gibt 'bacon local' aus

eggs = 'global' ❸
bacon()
print(eggs) # gibt 'global' aus
```

Wenn Sie dieses Programm ausführen, erhalten Sie folgende Ausgabe:

```
bacon local  
spam local  
bacon local  
global
```

Tatsächlich enthält dieses Programm drei verschiedene Variablen, die aber verwirrenderweise alle den Namen eggs haben. Es handelt sich dabei um folgende Variablen:

1. Die Variable eggs im lokalen Gültigkeitsbereich des Aufrufs von `spam()` (1).
2. Die Variable eggs im lokalen Gültigkeitsbereich des Aufrufs von `bacon()` (2).
3. Die Variable eggs im globalen Gültigkeitsbereich (3).

Da diese drei Variablen alle denselben Namen haben, kann es ziemlich kompliziert werden, nachzuvollziehen, welche zu einem bestimmten Zeitpunkt gerade verwendet wird. Daher sollten Sie es vermeiden, Variablen in unterschiedlichen Gültigkeitsbereichen denselben Namen zu geben.

## Die Anweisung `global`

Wenn Sie innerhalb einer Funktion eine globale Variable bearbeiten wollen, müssen Sie die Anweisung `global` verwenden. Eine Zeile wie `global eggs` am Anfang einer Funktion weist Python an: »In dieser Funktion bezieht sich eggs auf die globale Variable, also erstelle keine lokale Variable mit diesem Namen!« Geben Sie als Beispiel den folgenden Code in den Dateieditor ein und speichern Sie ihn als `sameName2.py`:

```
def spam():  
    global eggs ❶  
    eggs = 'spam' ❷  
  
    eggs = 'global'  
    spam()  
    print(eggs)
```

Wenn Sie dieses Programm ausführen, gibt der letzte Aufruf von `print()` Folgendes aus:

```
spam
```

Da `eggs` am Anfang von `spam()` als global deklariert ist (❶), erfolgt die Zuweisung von '`spam`' zu `eggs` (❷) an der globalen Variablen `eggs`. Es wird keine lokale Variable namens `eggs` erstellt.

Es gibt vier Regeln, mit deren Hilfe Sie lokale und globale Variablen unterscheiden können:

1. Wenn eine Variable in einem globalen Gültigkeitsbereich verwendet wird (also außerhalb irgendeiner Funktion), dann handelt es sich um eine globale Variable.
2. Wird in einer Funktion die Anweisung `global` für die Variable verwendet, handelt es sich um eine globale Variable.
3. Wird die Variable anderenfalls in einer Zuweisungsanweisung innerhalb der Funktion verwendet, handelt es sich um eine lokale Variable.
4. Wird die Variable dagegen nicht in einer Zuweisungsanweisung verwendet, handelt es sich um eine globale Variable.

Um ein besseres Gefühl für diese Regeln zu bekommen, schauen Sie sich das folgende Beispielprogramm an. Geben Sie den folgenden Code in den Dateieditor ein und speichern Sie ihn als `sameName3.py`:

```
def spam():
    global eggs ❶
    eggs = 'spam' # globale Variable

def bacon():
    eggs = 'bacon' # lokale Variable ❷

def ham():
    print(eggs) # globale Variable ❸

eggs = 42 # globale Variable
spam()
print(eggs)
```

In der Funktion `spam()` ist `eggs` die globale Variable dieses Namens, da zu Anfang die Anweisung `global` steht (❶). Dagegen ist `eggs` in `bacon()` eine lokale Variable, da es in dieser Funktion eine Zuweisungsanweisung für sie gibt (❷). In `ham()` ist `eggs` jedoch wiederum die globale Variable, da es in dieser Funktion zwar keine `global`-Anweisung, aber auch keine Zuweisungsanweisung gibt. Wenn Sie `sameName3.py` ausführen, erhalten Sie folgende Ausgabe:

```
spam
```

Innerhalb einer Funktion ist eine Variable entweder global oder lokal. Es ist nicht möglich, im Code einer Funktion sowohl eine lokale als auch eine globale Variable mit demselben Namen zu verwenden.

### Hinweis

Wenn Sie den Wert einer globalen Variablen von einer Funktion aus ändern wollen, so müssen Sie für diese Variable die Anweisung `global` verwenden.

Wenn Sie wie in dem folgenden Programm versuchen, eine lokale Variable in einer Funktion zu verwenden, bevor Sie ihr einen Wert zuweisen, gibt Python eine Fehlermeldung aus. Um sich das anzusehen, geben Sie den folgenden Code in den Dateieditor ein und speichern Sie ihn als *sameName4.py*:

```
def spam():
    print(eggs) # FEHLER!
    eggs = 'spam local' ❶

eggs = 'global' ❷
spam()
```

Wenn Sie dieses Programm ausführen, erhalten Sie folgende Fehlermeldung:

```
Traceback (most recent call last):
  File "C:/test3784.py", line 6, in <module>
    spam()
  File "C:/test3784.py", line 2, in spam
    print(eggs) # ERROR!
UnboundLocalError: local variable 'eggs' referenced before assignment
```

Der Grund für diesen Fehler ist, dass Python die Zuweisungsanweisung für `eggs` in der Funktion `spam()` sieht (❶) und daher davon ausgeht, dass `eggs` eine lokale Variable ist. Allerdings wird `print(eggs)` ausgeführt, bevor `eggs` irgendein Wert zugewiesen wurde, also zu einem Zeitpunkt, an dem die lokale Variable `eggs` noch gar nicht existiert (❷). In einem solchen Fall greift Python *nicht* auf die globale Variable `eggs` zurück.

### Funktionen als »Blackbox«

Meistens müssen Sie von einer Funktion nur die Eingaben (Parameter) und die Ausgabewerte kennen. Es ist oft gar nicht nötig, sich mit der Frage zu belasten, was der Code der Funktion im Einzelnen macht. Wenn Sie Funktionen von dieser hohen Warte aus betrachten, behandeln Sie sie wie eine Blackbox.

Das ist ein grundlegendes Prinzip in der modernen Programmierung. Weiter hinten in diesem Buch werden Sie einige Module mit Funktionen kennenlernen, die von anderen Personen geschrieben wurden. Wenn Sie neugierig sind, können Sie sich zwar auch den Quellcode ansehen, doch um diese Funktionen nutzen zu können, müssen Sie deren internen Mechanismus nicht kennen. Da empfohlen wird, Funktionen ohne globale Variablen zu schreiben, müssen Sie sich gewöhnlich auch keine Sorgen darüber machen, dass sich der Code der Funktion auf den Rest Ihres Programms auswirkt.

## Ausnahmebehandlung

Bis jetzt hat das Auftreten einer Fehlermeldung oder einer *Ausnahme* in Ihren Python-Programmen bedeutet, dass das gesamte Programm abstürzt. In Produktionsprogrammen wollen Sie das natürlich nicht. Stattdessen sollte das Programm in der Lage sein, Fehler zu erkennen, damit umzugehen und dann weiterzulaufen.

Betrachten Sie beispielsweise das folgende Programm, bei dem eine Division durch null vorkommen kann. Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *zeroDivide.py*:

```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Hier definieren wir die Funktion `spam` mit einem Parameter und geben dann die Werte dieser Funktion für verschiedene Parameter aus, um zu sehen, was geschieht. Wenn Sie den vorstehenden Code ausführen, erhalten Sie folgende Ausgabe:

```
21.0
3.5
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

Der Fehler `ZeroDivisionError` tritt auf, wenn Sie versuchen, eine Zahl durch null zu teilen. An der Zeilennummer in der Fehlermeldung können Sie ablesen, welche `return`-Anweisung in `spam()` den Fehler verursacht hat.

Fehler lassen sich mit den Anweisungen `try` und `except` handhaben. Stellen Sie Code, der einen Fehler verursachen kann, in eine `try`-Klausel. Wenn tatsächlich ein Fehler auftritt, rückt die Programmausführung zum Beginn der anschließenden `except`-Klausel vor.

In unserem Beispiel können Sie also den Code für die Division in eine `try`-Klausel stellen und in einer `except`-Klausel Code für den Fall angeben, dass eine Division durch null auftritt:

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Wenn Code in einer `try`-Klausel einen Fehler hervorruft, fährt die Programmausführung unmittelbar mit dem Code in der `except`-Klausel fort und läuft anschließend ganz normal weiter. Der vorstehende Code führt zu folgender Ausgabe:

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

Wenn in einem `try`-Block Funktionsaufrufe stehen, werden auch jegliche Fehler abgefangen, die dabei auftreten. Betrachten Sie das folgende Programm, bei dem die Aufrufe von `spam()` in einem `try`-Block stehen:

```
def spam(divideBy):  
    return 42 / divideBy  
  
try:  
    print(spam(2))  
    print(spam(12))  
    print(spam(0))  
    print(spam(1))  
except ZeroDivisionError:  
    print('Error: Invalid argument.')
```

Dieses Programm führt zu folgender Ausgabe:

```
21.0  
3.5  
Error: Invalid argument.
```

In diesem Fall wird `print(spam(1))` niemals ausgeführt, da die Ausführung nach dem Code in der `except`-Klausel nicht wieder in den `try`-Block zurückspringt, sondern ganz normal fortgesetzt wird.

## Ein kurzes Programm: Zahlen raten

Die primitiven Beispiele, die ich Ihnen bis jetzt vorgeführt habe, mögen vielleicht ganz nützlich sein, um einfache Prinzipien zu veranschaulichen, aber im Folgenden wollen wir uns ansehen, wie wir alles Gelernte in einem Programm zusammenfassen, das auch wirklich etwas tut. Dazu verwenden wir ein einfaches Zahlenratespiel. Wenn Sie das Programm ausführen, erhalten Sie eine Ausgabe wie die folgende:

```
I am thinking of a number between 1 and 20.  
Take a guess.  
10  
Your guess is too low.  
Take a guess.  
15  
Your guess is too low.  
Take a guess.  
17  
Your guess is too high.  
Take a guess.  
16  
Good job! You guessed my number in 4 guesses!
```

Geben Sie den folgenden Quellcode in den Dateieditor ein und speichern Sie ihn als `guessTheNumber.py`:

```
# Dies ist ein Zahlenratespiel.  
import random  
secretNumber = random.randint(1, 20)  
print('I am thinking of a number between 1 and 20.')  
  
# Fordert den Spieler sechsmal zum Raten der Zahl auf.  
for guessesTaken in range(1, 7):  
    print('Take a guess.')  
    guess = int(input())  
  
    if guess < secretNumber:  
        print('Your guess is too low.')  
    elif guess > secretNumber:  
        print('Your guess is too high.')  
    else:  
        break # Diese Bedingung tritt ein, wenn die Zahl erraten wurde!  
  
if guess == secretNumber:  
    print('Good job! You guessed my number in ' + str(guessesTaken)  
          + ' guesses!')  
else:  
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

Sehen wir uns diesen Code nun Zeile für Zeile an. Als Erstes haben wir Folgendes:

```
# Dies ist ein Zahlenratespiel.  
import random  
secretNumber = random.randint(1, 20)
```

Der Kommentar am Anfang des Codes erklärt zunächst einmal, wozu das Programm da ist. Danach importiert das Programm das Modul `random`, damit es mit der Funktion `random.randint()` die zu ratende Zahl generieren kann. Der Rückgabewert, ein zufälliger Integer zwischen 1 und 20, wird in der Variablen `secretNumber` gespeichert.

```
print('I am thinking of a number between 1 and 20.')  
  
# Fordert den Spieler sechsmal zum Raten der Zahl auf.  
for guessesTaken in range(1, 7):  
    print('Take a guess.')  
    guess = int(input())
```

Das Programm teilt dem Spieler nun mit, dass es sich eine Geheimzahl ausgedacht hat und den Spieler sechsmal raten lässt. Der Code, der den Spieler einen Rateversuch eingeben lässt und die Zahl anschließend überprüft, steht in einer `for`-Schleife, die maximal sechsmal ausgeführt wird. In der Schleife wird als Erstes die Eingabe des Spielers entgegengenommen. Da `input()` einen String zurückgibt, wird dieser Rückgabewert sofort an die Funktion `int()` übergeben, die ihn in einen Integerwert umwandelt. Dieser wird schließlich in der Variablen `guess` abgelegt.

```

if guess < secretNumber:
    print('Your guess is too low.')
elif guess > secretNumber:
    print('Your guess is too high.')

```

Diese wenigen Codezeilen prüfen, ob die geratene Zahl größer oder kleiner als die Geheimzahl ist. In beiden Fällen wird auf dem Bildschirm ein entsprechender Tipp ausgegeben.

```

else:
    break # Diese Bedingung tritt ein, wenn die Zahl erraten wurde!

```

Ist die geratene Zahl weder kleiner noch größer als die Geheimzahl, so muss sie gleich dieser Zahl sein. In diesem Fall wird die for-Schleife abgebrochen.

```

if guess == secretNumber:
    print('Good job! You guessed my number in ' + str(guessesTaken) + ' guesses!')
else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))

```

Nach der for-Schleife prüft die vorstehende if...else-Anweisung, ob der Spieler die Zahl erraten hat, und gibt eine entsprechende Meldung auf dem Bildschirm aus. In jedem Fall zeigt das Programm eine Variable mit einem Integerwert an (guessesTaken bzw. secretNumber). Da diese Integerwerte mit Strings verkettet werden müssen, werden die Variablen an die Funktion str() übergeben, die die Stringversionen der Integer zurückgibt. Jetzt können die Strings mit dem Operator + verkettet und schließlich an print() übergeben werden.

## Zusammenfassung

Funktionen sind die wichtigste Möglichkeit, um Ihren Code logisch zu gruppieren. Da Variablen in Funktionen eigene lokale Gültigkeitsbereiche haben, kann der Code in einer Funktion die Werte von Variablen in anderen Funktionen nicht unmittelbar beeinflussen. Das schränkt den Umfang des Codes ein, der die Werte von Variablen ändern kann, was sich wiederum beim Debugging als sehr hilfreich erweist.

Funktionen bilden eine hervorragende Möglichkeit, um Ihren Code zu gliedern. Betrachten Sie sie als Blackbox: Sie haben Eingaben in der Form von Parametern und Ausgaben in Form von Rückgabewerten, aber ihr interner Code wirkt sich nicht auf die Variablen in anderen Funktionen aus.

In den vorherigen Kapiteln führte ein einziger Fehler zum Absturz des gesamten Programms. In diesem Kapitel haben Sie die Anweisungen `try` und `except` kennengelernt, mit denen besonderer Code ausgeführt werden kann, wenn ein Fehler erkannt wurde. Das macht Ihren Code widerstandsfähiger gegen häufig auftretende Fehler.

## Wiederholungsfragen

1. Warum ist es von Vorteil, in Ihren Programmen Funktionen zu verwenden?
2. Wann wird der Code in einer Funktion ausgeführt – bei der Definition oder beim Aufruf der Funktion?
3. Mit welcher Anweisung erstellen Sie eine Funktion?
4. Was ist der Unterschied zwischen einer Funktion und einem Funktionsaufruf?
5. Wie viele globale Gültigkeitsbereiche hat ein Python-Programm? Wie viele lokale Gültigkeitsbereiche?
6. Was geschieht mit den Variablen in einem lokalen Gültigkeitsbereich, wenn der Funktionsaufruf die Steuerung zurückgibt?
7. Was ist ein Rückgabewert? Kann ein Rückgabewert Teil eines Ausdrucks sein?
8. Welchen Rückgabewert hat der Aufruf einer Funktion, die keine return-Anweisung hat?
9. Wie können Sie dafür sorgen, dass in einer Funktion eine globale Variable verwendet wird?
10. Welchen Datentyp hat `None`?
11. Was macht die Anweisung `import areallyourpetsnamederic`?
12. Wie rufen Sie die Funktion `bacon()` aus dem Modul `spam` auf, nachdem Sie `spam` importiert haben?
13. Wie verhindern Sie, dass ein Programm beim Auftreten eines Fehlers abstürzt?
14. Was steht in einer `try`-Klausel und was in der `except`-Klausel?

## Übungsprojekte

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben lösen:

### Die Collatz-Folge

Schreiben Sie die Funktion `collatz()` mit dem Parameter `number`. Wenn `number` eine gerade Zahl ist, soll diese Funktion `number // 2` ausgeben und zurückgeben. Ist `number` dagegen ungerade, soll `collatz` das Ergebnis von `3 * number + 1` ausgeben und zurückgeben.

Schreiben Sie anschließend ein Programm, das eine ganze Zahl als Benutzeingebe entgegennimmt und anschließend für diese Zahl und danach für die Rückgabewerte immer wieder `collatz()` aufruft, bis die Funktion den Wert 1 zurückgibt. (Erstaunlicherweise funktioniert das bei jeder beliebigen Ganzzahl als Ausgangspunkt. Früher oder später erreichen Sie mit dieser Folge den Wert 1. Selbst Mathematiker wissen nicht, warum das so ist. Dieses Programm veranschaulicht die sogenannte *Collatz-Folge*, die manchmal auch als »das einfachste unlösbare Problem der Mathematik« bezeichnet wird.)

Vergessen Sie nicht, den Rückgabewert von `input()` mithilfe von `int()` in einen Integer umzuwandeln, denn zunächst einmal ist er ein String.

Tipp: Der Integer `number` ist gerade, wenn `number % 2 == 0`, und ungerade, wenn `number % 2 == 1`.

Die Ausgabe dieses Programms sieht wie folgt aus:

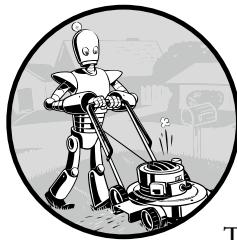
```
Enter number:  
3  
10  
5  
16  
8  
4  
2  
1
```

## Eingabevalidierung

Ergänzen Sie das vorherige Projekt um `try-` und `except-`Anweisungen, um festzustellen, ob der Benutzer etwas anderes als einen Integer eingegeben hat. Normalerweise gibt die Funktion `int()` den Fehler `ValueError` zurück, wenn ihr etwas anderes als ein Integer übergeben wird, z. B. `int('puppy')`. Geben Sie in der `except`-Klausel eine Meldung aus, die den Benutzer darauf hinweist, dass er ganze Zahlen eingeben muss.

# 4

## Listen



Bevor Sie ernsthaft damit beginnen können, Programme zu schreiben, müssen Sie sich noch mit einem weiteren Thema beschäftigen, nämlich mit den miteinander verwandten Datentypen für Listen und Tupel. Beide können mehrere Werte enthalten, was es einfacher macht,

Programme zu schreiben, die mit großen Datenmengen umgehen. Da Listen wiederum andere Listen einschließen können, ist es damit auch möglich, Daten hierarchisch zu gliedern.

In diesem Kapitel beschäftigen wir uns mit den Grundlagen von Listen. Außerdem stelle ich *Methoden* vor, also Funktionen, die an die Werte eines bestimmten Datentyps gebunden sind. Des Weiteren werde ich kurz auf die listenähnlichen Datentypen der Tupel und Strings eingehen und diese mit Listenwerten vergleichen. Im nächsten Kapitel geht es dann um den Datentyp der Dictionarys (Wörterbücher).

### Der Datentyp für Listen

Eine *Liste* ist ein Wert, der mehrere Werte in einer geordneten Reihenfolge enthält, etwa wie folgt: `['cat', 'bat', 'rat', 'elephant']`. So wie Anfang und Ende von

Stringwerten durch Anführungszeichen gekennzeichnet sind, so beginnen und enden Listen mit eckigen Klammern. Der Begriff *Listenwert* bezeichnet dabei die Liste selbst (da sie wie jeder andere Wert in einer Variablen gespeichert oder einer Funktion übergeben werden kann), nicht auf die darin enthaltenen Werte. Ein Listenwert sieht wie folgt aus: Die einzelnen Werte innerhalb der Liste werden *Elemente* genannt und durch Kommata getrennt. Zur Veranschaulichung geben Sie Folgendes in die interaktive Shell ein:

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
>>> spam = ['cat', 'bat', 'rat', 'elephant'] ❶
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

Der Variable `spam` wird nur ein einziger Wert zugewiesen, nämlich der Listenwert (❶), der aber selbst wiederum mehrere andere Werte enthält.

Übrigens gibt es vergleichbar zum leeren String '' auch eine leere Liste [], die keinerlei Elemente aufweist.

### Einzelne Elemente aus einer Liste mithilfe des Index abrufen

Wenn Sie die Liste ['cat', 'bat', 'rat', 'elephant'] in der Variable `spam` gespeichert haben, wird `spam[0]` zu 'cat' ausgewertet, `spam[1]` zu 'bat' usw. Der Integer in den eckigen Klammern, der die Position in der Liste angibt, wird als *Index* bezeichnet. Das erste Element in einer Liste hat den Index 0, das zweite den Index 1, das dritte den Index 2 usw. Abb. 4–1 zeigt den Listenwert, der `spam` zugeordnet ist, und gibt an, wozu die einzelnen Indexausdrücke ausgewertet werden.

```
spam = ["cat", "bat", "rat", "elephant"]
      ↑   ↑   ↑   ↑
spam[0] spam[1] spam[2] spam[3]
```

**Abb. 4–1** Ein Listenwert in der Variablen `spam`. Die Pfeile zeigen, auf welche Elemente sich die Indizes jeweils beziehen.

Geben Sie zur Veranschaulichung die folgenden Ausdrücke in die interaktive Shell ein. Zu Anfang weisen Sie der Variablen `spam` eine Liste zu.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
'rat'
>>> spam[3]
'elephant'
>>> ['cat', 'bat', 'rat', 'elephant'][3]
'elephant'
>>> 'Hello ' + spam[0]    ❶
'Hello cat'    ❷
>>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
'The bat ate the cat.'
```

Der Ausdruck 'Hello ' + spam[0] (❶) wird dabei zu 'Hello ' + 'cat' ausgewertet, da spam[0] den String 'cat' bezeichnet. Dieser Ausdruck wiederum ergibt den Stringwert 'Hello cat' (❷).

Wenn Sie einen Index verwenden, der über die Anzahl der Elemente in der Liste hinausgeht, gibt Python die Fehlermeldung `IndexError` aus:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    spam[10000]
IndexError: list index out of range
```

Indizes können immer nur Integerwerte sein, keine Fließkommazahlen. Das folgende Beispiel ruft den Fehler `TypeError` hervor:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    spam[1.0]
TypeError: list indices must be integers, not float
>>> spam[int(1.0)]
'bat'
```

Listen können auch andere Listen enthalten. Um auf die Elemente zuzugreifen, verwenden Sie Mehrfachindizes:

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
>>> spam[0][1]
'bat'
>>> spam[1][4]
50
```

Der erste Index gibt an, welcher Listenwert abgerufen werden soll, während der zweite das Element in diesem Listenwert bezeichnet. Beispielsweise führt `spam[0]` [1] zu 'bat', dem zweiten Element in der ersten Liste. Wenn Sie nur einen Index angeben, gibt das Programm den kompletten Listenwert an dieser Indexposition aus.

## Negative Indizes

Die Indexzahlen verlaufen von null aufwärts, doch Sie können auch negative Integer als Index angeben. Dabei bezeichnet der Wert -1 den letzten Index in der Liste, -2 den vorletzten usw. Geben Sie folgenden Code in die interaktive Shell ein:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
'The elephant is afraid of the bat.'
```

## Teillisten mit Slices abrufen

Mit einem Index können Sie ein einzelnes Element aus einer Liste abrufen, mit einem *Slice* dagegen mehrere Elemente in Form einer neuen Liste. Wie ein Index wird ein Slice in eckigen Klammern angegeben, allerdings in Form von zwei durch einen Doppelpunkt getrennten Integern. Beachten Sie den Unterschied:

- `spam[2]` bezeichnet einen Index in einer Liste (ein Integer).
- `spam[1:4]` bezeichnet einen Slice in einer Liste (zwei Integer).

Der erste Integer in einem Slice gibt die Indexposition an, bei der der Slice beginnt, der zweite diejenige, bei der er endet. Der Slice enthält alle Werte vom ersten bis zum letzten angegebenen Index, allerdings ohne diesen letzten Index. Ergebnis eines Slice ist ein neuer Listenwert. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

Zur Verkürzung der Schreibweise können Sie auch auf einen oder beide Indizes auf den Seiten des Doppelpunkts verzichten. Wenn Sie den ersten Index weglassen, wird 0 angenommen, also der Beginn der Liste, wenn Sie den zweiten Index weglassen, das Ende der Liste. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
['cat', 'bat']
>>> spam[1:]
['bat', 'rat', 'elephant']
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

### Die Länge einer Liste mit len() abrufen

Die Funktion `len()` kann nicht nur die Anzahl der Zeichen in einem Stringwert nennen, sondern auch die Anzahl der Elemente in dem ihr übergebenen Listenwert. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

### Werte in einer Liste anhand des Index ändern

Auf der linken Seite einer Zuweisungsanweisung steht normalerweise ein Variablenname, etwa wie in `spam = 42`. Sie können jedoch auch einen Listenindex angeben, um den Wert an diesem Index zu ändern. Beispielsweise bedeutet `spam[1] = 'aardvark'`: »Weise dem Element an Index 1 der Liste `spam` den String 'aardvark' zu.« Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

## Listenverkettung und -wiederholung

Ebenso wie der Operator + zwei Strings zu einem neuen zusammenbaut, kann er auch zwei Listen zu einem neuen Listenwert verketten. Der Operator \* kann auch mit einer Liste und einem Integerwert verwendet werden, um die Liste zu wiederholen. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

## Elemente mit del aus einer Liste entfernen

Die Anweisung `del` löscht die Werte am angegebenen Listenindex, woraufhin alle nachfolgenden Werte einen Indexplatz aufrücken. Geben Sie beispielsweise Folgendes in die interaktive Shell ein:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

Die Anweisung `del` können Sie auch verwenden, um eine einfache Variable zu löschen, also praktisch wie eine Anweisung zum »Aufheben der Zuweisung«. Wenn Sie versuchen, auf die Variable zuzugreifen, nachdem Sie sie gelöscht haben, erhalten Sie die Fehlermeldung `NameError`, da die Variable nicht mehr existiert.

In der Praxis müssen Sie einfache Variablen jedoch so gut wie nie löschen. Die Anweisung `del` wird meistens verwendet, um Elemente aus Listen zu entfernen.

## Listen verwenden

Einsteiger in die Programmierung sind oft versucht, viele einzelne Variablen zu erstellen, um Gruppen ähnlicher Werte zu speichern. Nehmen wir an, ich möchte die Namen meiner Katzen festhalten. Dann könnte ich Code wie den folgenden schreiben:

```
catName1 = 'Zophie'  
catName2 = 'Pooka'  
catName3 = 'Simon'  
catName4 = 'Lady Macbeth'  
catName5 = 'Fat-tail'  
catName6 = 'Miss Cleo'
```

(Ich möchte allerdings darauf hinweisen, dass ich in Wirklichkeit nicht so viele Katzen habe!) Das ist jedoch eine schlechte Vorgehensweise. Erstens kann das Programm niemals mehr Katzennamen speichern, als Variablen zur Verfügung stehen, was zu einem Problem führt, wenn sich die Anzahl der Katzen ändert. Bei Programmen, die auf diese Weise geschrieben sind, gibt es gewöhnlich auch eine Menge an doppeltem oder fast identischem Code. Schauen Sie einmal nach, wie viel doppelten Code es in dem folgenden Programm namens *allMyCats1.py* gibt:

```
print('Enter the name of cat 1:')  
catName1 = input()  
print('Enter the name of cat 2:')  
catName2 = input()  
print('Enter the name of cat 3:')  
catName3 = input()  
print('Enter the name of cat 4:')  
catName4 = input()  
print('Enter the name of cat 5:')  
catName5 = input()  
print('Enter the name of cat 6:')  
catName6 = input()  
print('The cat names are:')  
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' + catName4 + ' ' +  
catName5 + ' ' + catName6)
```

Anstatt mehrerer ähnlicher Variablen können Sie auch eine einzige Variable mit einem Listenwert verwenden. Die folgende verbesserte Version von *allMyCats1.py* nutzt eine einzige Liste und kann darin so viele Katzennamen speichern, wie der Benutzer eingibt. Geben Sie den folgenden Quellcode in ein Dateieditorfenster ein und speichern Sie ihn als *allMyCasts2.py*:

```
catNames = []  
while True:  
    print('Enter the name of cat ' + str(len(catNames) + 1) +  
          ' (Or enter nothing to stop.):')  
    name = input()  
    if name == '':  
        break  
    catNames = catNames + [name] # list concatenation  
print('The cat names are:')  
for name in catNames:  
    print(' ' + name)
```

Wenn Sie dieses Programm ausführen, erhalten Sie folgende Ausgabe:

```
Enter the name of cat 1 (Or enter nothing to stop.):  
Zophie  
Enter the name of cat 2 (Or enter nothing to stop.):  
Pooka  
Enter the name of cat 3 (Or enter nothing to stop.):  
Simon  
Enter the name of cat 4 (Or enter nothing to stop.):  
Lady Macbeth  
Enter the name of cat 5 (Or enter nothing to stop.):  
Fat-tail  
Enter the name of cat 6 (Or enter nothing to stop.):  
Miss Cleo  
Enter the name of cat 7 (Or enter nothing to stop.):  
  
The cat names are:  
Zophie  
Pooka  
Simon  
Lady Macbeth  
Fat-tail  
Miss Cleo
```

Der Vorteil der Verwendung einer Liste besteht darin, dass sich die Daten jetzt in einer Struktur befinden, sodass das Programm die Daten auf viel flexiblere Art und Weise verarbeiten kann, als wenn sie in mehreren ähnlichen Variablen steckten.

### For-Loops für Listen

In Kapitel 2 haben Sie gelernt, wie Sie einen Codeblock mithilfe einer `for`-Schleife eine gegebene Anzahl von Malen ausführen. Technisch gesehen wiederholt eine `for`-Schleife den Codeblock einmal für jeden Wert in einer Liste oder einem listenähnlichen Wert. Nehmen wir an, Sie führen folgenden Code aus:

```
for i in range(4):  
    print(i)
```

Die Ausgabe dieses Programms lautet wie folgt:

```
0  
1  
2  
3
```

Der Grund dafür ist, dass der Rückgabewert von `range(4)` ein listenähnlicher Wert ist, den Python ähnlich wie `[0, 1, 2, 3]` auffasst. Das folgende Programm führt zu der gleichen Ausgabe:

```
for i in [0, 1, 2, 3]:  
    print(i)
```

Diese `for`-Schleife durchläuft eine Klausel, wobei die Variable `i` bei jeder Iteration auf den nächstfolgenden Wert der Liste `[0, 1, 2, 3]` gesetzt wird.

### Hinweis

In diesem Buch verwende ich den Begriff *listenähnlich* für Datentypen, die technisch als *Sequenzen* oder *Folgen* bezeichnet werden. Die technische Definition dieses Begriffs müssen Sie jedoch nicht kennen.

Eine übliche Python-Technik besteht darin, `range(len(eineListe))` mit einer `for`-Schleife zu verwenden, um über die Indizes einer Liste zu iterieren. Geben Sie beispielsweise folgenden Code in die interaktive Shell ein:

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']  
>>> for i in range(len(supplies)):  
    print('Index ' + str(i) + ' in supplies is: ' + supplies[i])  
Index 0 in supplies is: pens  
Index 1 in supplies is: staplers  
Index 2 in supplies is: flame-throwers  
Index 3 in supplies is: binders
```

Die Verwendung von `range(len(supplies))` in dieser `for`-Schleife ist ziemlich praktisch, da der Code in der Schleife auf den Index (die Variable `i`) zugreifen und den Wert an diesem Index (also `supplies[i]`) lesen kann. Das Beste aber ist, dass `range(len(supplies))` stets sämtliche Indizes von `supplies` durchläuft, wie viele Elemente die Liste auch immer enthält.

### Die Operatoren `in` und `not in`

Mit den Operatoren `in` und `not in` können Sie feststellen, ob sich ein Wert in einer Liste befindet oder nicht. Wie die anderen Operatoren werden auch `in` und `not in` in Ausdrücken verwendet und sie verknüpfen zwei Werte: den Wert, nach dem in der Liste gesucht werden soll, und die betreffende Liste. Ausgewertet werden diese Ausdrücke zu einem booleschen Wert. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

In dem folgenden Beispielprogramm gibt der Benutzer den Namen eines Haustiers ein, worauf geprüft wird, ob dieser Name auf einer Liste von Haustieren steht. Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *myPets.py*:

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter a pet name:')
name = input()
if name not in myPets:
    print('I do not have a pet named ' + name)
else:
    print(name + ' is my pet.')
```

Die Ausgabe sieht wie folgt aus:

```
Enter a pet name:
Footfoot
I do not have a pet named Footfoot
```

### **Der Trick mit der Mehrfachzuweisung**

Die *Mehrfachzuweisung* ist eine Abkürzung, mit der Sie in einer einzigen Codezeile mehreren Variablen Werte aus einer Liste zuweisen können. Wenn Sie das einzeln tun wollten, müssten Sie wie folgt vorgehen:

```
>>> cat = ['fat', 'black', 'loud']
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

Stattdessen aber können Sie einfach Folgendes schreiben:

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition = cat
```

Dabei muss es jedoch genauso viele Variablen wie Elemente in der Liste geben. Andernfalls meldet Python den Fehler `ValueError`:

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition, name = cat
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    size, color, disposition, name = cat
ValueError: need more than 3 values to unpack
```

## Erweiterte Zuweisungsoperatoren

Nachdem Sie einer Variablen einen Wert zugewiesen haben, werden Sie sie im weiteren Verlauf gewöhnlich noch oft verwenden. Im folgenden Beispiel wird der Variablen `spam` zunächst der Wert 42 zugewiesen, der dann später um 1 erhöht wird:

```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```

Als Kurzschreibweise können Sie dazu auch den erweiterten Zuweisungsoperator `+=` verwenden:

```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

Wie Sie in Tabelle 4–1 sehen, gibt es erweiterte Zuweisungsoperatoren für `+`, `-`, `*`, `/` und `%`:

Zuweisungsanweisung	Entsprechende Anweisung mit erweitertem Zuweisungsoperator
<code>spam = spam + 1</code>	<code>spam += 1</code>
<code>spam = spam - 1</code>	<code>spam -= 1</code>
<code>spam = spam * 1</code>	<code>spam *= 1</code>
<code>spam = spam / 1</code>	<code>spam /= 1</code>
<code>spam = spam % 1</code>	<code>spam %= 1</code>

**Tab. 4–1** Erweiterte Zuweisungsoperatoren

Der Operator `+=` kann auch für die String- und Listenverkettung eingesetzt werden, der Operator `*=` auch für String- und Listenwiederholung. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = 'Hello'
>>> spam += ' world!'
>>> spam
'Hello world!'
>>> bacon = ['Zophie']
>>> bacon *= 3
>>> bacon
['Zophie', 'Zophie', 'Zophie']
```

## Methoden

Eine *Methode* ist eine Funktion, die für einen Wert aufgerufen wird. Wenn wir beispielsweise einen Listenwert in `spam` gespeichert haben, können wir die Listenmethode `index()` (die ich im Folgenden erklären werde) für diese Liste wie folgt aufrufen: `spam.index('hello')`.

Jeder Datentyp verfügt über seinen eigenen Satz von Methoden. So hat der Listendatentyp beispielsweise einige sehr nützliche Methoden, um Elemente in einer Liste zu finden, hinzuzufügen, zu entfernen oder auf andere Weise zu bearbeiten.

### Elemente in einer Liste mit der Methode `index()` finden

Für Listenwerte gibt es die Methode `index()`. Sie nimmt einen Wert entgegen und wenn dieser Wert in der Liste vorhanden ist, so wird dessen Index zurückgegeben. Gibt es den betreffenden Wert in der Liste nicht, gibt Python den Fehler `ValueError` aus. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

Kommt ein Wert mehrfach in einer Liste vor, so wird der Index des ersten Auftretens zurückgegeben. Wenn Sie Folgendes in die interaktive Shell eingeben, gibt `index()` nicht 3 zurück, sondern 1:

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

## Elemente mit den Methoden `append()` und `insert()` zu Listen hinzufügen

Um zu einer Liste neue Elemente hinzuzufügen, verwenden Sie die Methoden `append()` und `insert()`. Im folgenden Beispiel wird die Methode `append()` auf den Listenwert in der Variablen `spam` angewendet:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

Wie Sie sehen, hängt die Methode `append()` ihr Argument an das Ende der Liste an. Mit `insert()` dagegen können Sie einen Wert an einer beliebigen Indexposition in der Liste einfügen. Das erste Argument von `insert()` ist der Index für den neuen Wert, das zweite der einzufügende Wert. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

Beachten Sie, dass die korrekte Schreibweise `spam.append('moose')` bzw. `spam.insert(1, 'chicken')` lautet und nicht etwa `spam = spam.append('moose')` oder `spam = spam.insert(1, 'chicken')`. Weder `append()` noch `insert()` geben den neuen Wert von `spam` zurück. (Der Rückgabewert beider Methoden ist in Wirklichkeit `None` und das ist nicht das, was Sie hier als neuen Variablenwert speichern wollen!) Stattdessen wird die Liste unmittelbar geändert. Mehr darüber erfahren Sie im Abschnitt »Veränderbare und unveränderbare Datentypen« weiter hinten in diesem Kapitel.

Methoden gehören jeweils zu einem einzigen Datentyp. So sind `append()` und `insert()` Listenmethoden, die nur für Listenwerte aufgerufen werden können, nicht aber für andere Werte wie etwa Strings oder Integer. Wenn Sie Folgendes in die interaktive Shell eingeben, erhalten Sie die Fehlermeldung `AttributeError`:

```

>>> eggs = 'hello'
>>> eggs.append('world')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    eggs.append('world')
AttributeError: 'str' object has no attribute 'append'
>>> bacon = 42
>>> bacon.insert(1, 'world')
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    bacon.insert(1, 'world')
AttributeError: 'int' object has no attribute 'insert'

```

## Elemente mit remove() aus Listen entfernen

Wenn Sie die Methode `remove()` für eine Liste aufrufen, übergeben Sie ihr den Wert, den Sie aus dieser Liste entfernen möchten. Geben Sie Folgendes in die interaktive Shell ein:

```

>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']

```

Wenn Sie versuchen, einen Wert aus einer Liste zu entfernen, der gar nicht darin vorhanden ist, erhalten Sie den Fehler `ValueError`, wie das folgende Beispiel zeigt:

```

>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    spam.remove('chicken')
ValueError: list.remove(x): x not in list

```

Kommt der betreffende Wert mehrmals in der Liste vor, so wird nur das erste Vorkommen entfernt:

```

>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
>>> spam.remove('cat')
>>> spam
['bat', 'rat', 'cat', 'hat', 'cat']

```

Wenn Sie den Index des Elements kennen, den Sie aus einer Liste löschen wollen, können Sie die Anweisung `del` verwenden; kennen Sie dagegen den Wert des Elements, so nutzen Sie zum Entfernen `remove()`.

## Elemente in einer Liste mit sort() sortieren

Listen von Zahlenwerten oder Strings lassen sich mit der Methode `sort()` sortieren. Betrachten Sie dazu das folgende Beispiel:

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

Wenn Sie `True` für das Schlüsselwortargument `reverse` übergeben, sortiert `sort()` die Elemente in umgekehrter Reihenfolge:

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

Bei der Verwendung von `sort()` müssen Sie drei Dinge beachten. Erstens sortiert `sort()` die Liste unmittelbar. Versuchen Sie also nicht, den Rückgabewert zu erfassen, indem Sie Code wie `spam = spam.sort()` schreiben.

Zweitens können Sie keine Listen notieren, in denen sowohl Zahlen als auch Strings vorkommen, da Python solche unterschiedlichen Werte nicht vergleichen kann. Bei dem folgenden Beispiel ergibt sich daher der Fehler `TypeError`:

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    spam.sort()
TypeError: unorderable types: str() < int()
```

Drittens verwendet `sort()` statt der normalen alphabetischen Sortierung eine ASCII-Sortierung. Das bedeutet, dass Großbuchstaben grundsätzlich vor Kleinbuchstaben kommen. Ein kleines `a` steht daher hinter dem großen `Z`! Das können Sie an folgendem Beispiel erkennen:

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

Wenn Sie die Werte in regulärer alphabetischer Reihenfolge sortieren müssen, übergeben Sie im Aufruf der Methode `sort()` den Wert `str.lower` für das Schlüsselwortargument `key`:

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

Dadurch behandelt `sort()` alle Elemente in der Liste so, als begännen sie mit einem Kleinbuchstaben, ohne sie aber tatsächlich zu ändern.

## Beispielprogramm: Magic 8 Ball unter Verwendung einer Liste

Mithilfe von Listen können Sie das Programm Magic 8 Ball aus dem vorherigen Kapitel auf eine viel elegantere Weise schreiben. Anstatt mehrere Zeilen nahezu identischer `elif`-Anweisungen zu verwenden, können Sie eine einzelne Liste aufstellen, mit der der Code arbeitet. Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *magic8Ball2.py*:

```
import random

messages = ['It is certain',
    'It is decidedly so',
    'Yes definitely',
    'Reply hazy try again',
    'Ask again later',
    'Concentrate and ask again',
    'My reply is no',
    'Outlook not so good',
    'Very doubtful']

print(messages[random.randint(0, len(messages) - 1)])
```

### Ausnahmen von den Einrückungsregeln in Python

In den meisten Fällen erkennt Python anhand der Tiefe der Einrückung, zu welchem Block eine Codezeile gehört. Es gibt jedoch einige Ausnahmen von dieser Regel. Eine davon sind Listen, die mehrere Zeilen einer Quellcode datei umfassen können. Dabei spielt die Einrückung keine Rolle, denn Python weiß, dass die Liste erst mit der schließenden eckigen Klammer endet. Daher könnten Sie Ihren Code auch wie folgt schreiben:

```
spam = ['apples',
        'oranges',
                    'bananas',
    'cats']
print(spam)
```

In der Praxis sollten Sie Ihren Code natürlich so schreiben, dass die Listen klar und übersichtlich aussehen, etwa wie die Liste der Meldungen im Programm Magic 8 Ball.

Es ist auch möglich, eine Anweisung auf mehrere Zeilen aufzuteilen, indem Sie am Ende das Zeilenfortsetzungssymbol \ verwenden. Es bedeutet: »Diese Anweisung geht in der nächsten Zeile weiter.« Die Einrückung der Zeile, die auf das Zeichen \ folgt, spielt dann keine Rolle mehr. Beispielsweise ist folgender Code in Python gültig:

```
print('Four score and seven ' + \
      'years ago...')
```

Diese Tricks sind praktisch, um lange Zeilen umzuarrangieren und Ihren Python-Code damit besser lesbar zu machen.

Wenn Sie dieses Programm ausführen, werden Sie feststellen, dass es genauso funktioniert wie die ursprüngliche Version *magic8Ball.py*.

Beachten Sie den Ausdruck `random.randint(0, len(messages) - 1)`, den Sie hier als Index für `messages` verwenden. Dadurch wird für den Index eine Zufallszahl generiert, die von der Größe von `messages` abhängig ist – d. h., Sie erhalten eine Zufallszahl zwischen 0 und dem Wert von `len(messages)-1`. Der Vorteil dieser Vorgehensweise besteht darin, dass Sie Elemente zur Liste `messages` hinzufügen und daraus entfernen können, ohne diese Codezeile zu ändern. Wenn Sie den Code später ändern, müssen Sie weniger Codezeilen anfassen, was auch weniger Gelegenheiten dafür bietet, Fehler einzubauen.

## Listenähnliche Typen: Strings und Tupel

Listen sind nicht die einzigen Datentypen, die eine geordnete Folge von Werten darstellen. So können Sie sich beispielsweise auch einen String als eine »Liste« von Textzeichen vorstellen. Viele der Dinge, die Sie mit Listen anstellen können, lassen sich auch mit Strings tun: Sie können sie indizieren, Sie können Slices daraus entnehmen, sie in for-Schleifen verwenden, ihre Größe mit `len()` bestimmen und die Operatoren `in` und `not in` anwenden. Geben Sie zur Veranschaulichung Folgendes in die interaktive Shell ein:

```
>>> name = 'Zophie'
>>> name[0]
'Z'
>>> name[-2]
'i'
>>> name[0:4]
'Zoph'
>>> 'Zo' in name
True
>>> 'z' in name
False
>>> 'p' not in name
False
>>> for i in name:
    print('* * * ' + i + ' * * *')

* * * Z * * *
* * * o * * *
* * * p * * *
* * * h * * *
* * * i * * *
* * * e * * *
```

## Veränderbare und unveränderbare Datentypen

Listen und Strings unterscheiden sich jedoch in einem sehr wichtigen Punkt: Ein Listenwert ist ein *veränderbarer* Datentyp, da Sie Elemente zu ihm hinzufügen, aus ihm entfernen und ändern können. Dagegen ist ein String *unveränderbar* – er lässt sich nicht ändern. Wenn Sie versuchen, ein einzelnes Zeichen in einem String neu zuzuweisen, erhalten Sie die Fehlermeldung `TypeError`, wie das folgende Beispiel zeigt:

```
>>> name = 'Zophie a cat'  
>>> name[7] = 'the'  
Traceback (most recent call last):  
  File "<pyshell#50>", line 1, in <module>  
    name[7] = 'the'  
TypeError: 'str' object does not support item assignment
```

Um einen String zu »ändern«, müssen Sie ihn zerschneiden und verketten, um einen *neuen* String aus Teilen des alten zu erstellen. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> name = 'Zophie a cat'  
>>> newName = name[0:7] + 'the' + name[8:12]  
>>> name  
'Zophie a cat'  
>>> newName  
'Zophie the cat'
```

Die Angaben [0:7] und [8:12] verweisen hier auf die Zeichen, die nicht ersetzt werden sollen. Da Strings unveränderbar sind, wird der ursprüngliche String 'Zophie a cat' nicht geändert.

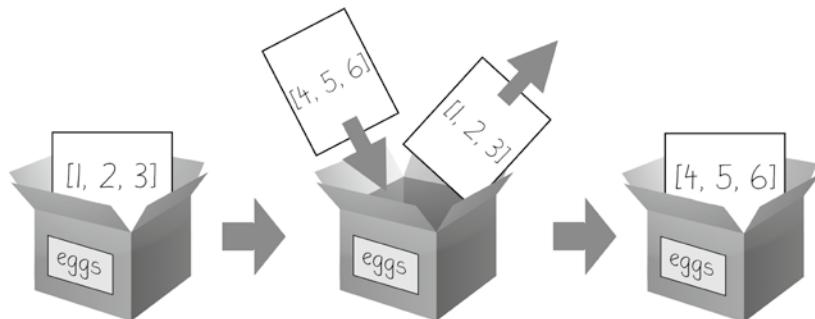
Ein Listenwert ist zwar veränderbar, doch die zweite Zeile im folgenden Code nimmt trotzdem keine Änderung an der Liste eggs vor:

```
>>> eggs = [1, 2, 3]  
>>> eggs = [4, 5, 6]  
>>> eggs  
[4, 5, 6]
```

Der Listenwert in eggs wird hier in Wirklichkeit nicht geändert. Stattdessen wird der alte Listenwert [1, 2, 3] komplett mit dem neuen Listenwert [4, 5, 6] überschrieben, wie Abbildung 4–2 deutlich zeigt.

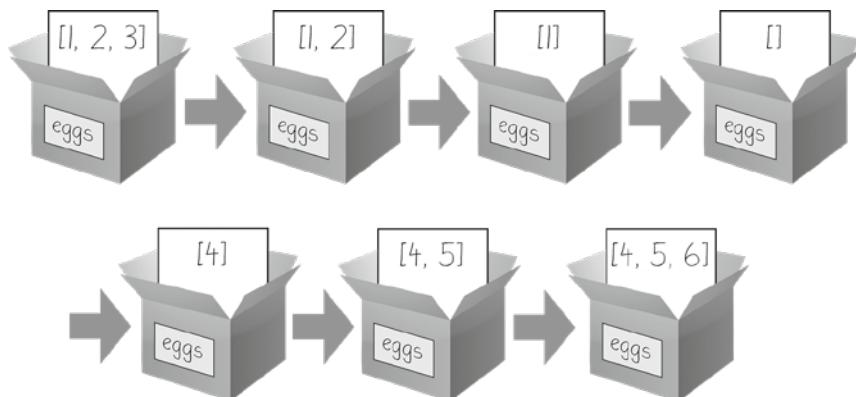
Wenn Sie die ursprüngliche Liste in eggs tatsächlich so ändern wollen, dass sie schließlich [4, 5, 6] lautet, so müssen Sie wie folgt vorgehen:

```
>>> eggs = [1, 2, 3]  
>>> del eggs[2]  
>>> del eggs[1]  
>>> del eggs[0]  
>>> eggs.append(4)  
>>> eggs.append(5)  
>>> eggs.append(6)  
>>> eggs  
[4, 5, 6]
```



**Abb. 4-2** Wenn `eggs = [4, 5, 6]` ausgeführt wird, so wird der Inhalt von `eggs` durch einen neuen Listenwert ersetzt.

In diesem Beispiel steht in `eggs` am Ende noch der gleiche Listenwert wie zu Anfang; er wurde lediglich geändert, aber nicht überschrieben. In Abb. 4-3 können Sie die sieben Schritte sehen, die die sieben ersten Zeilen dieses Beispiels ausführen.



**Abb. 4-3** Die Anweisung `del` und die Methode `append()` ändern den Listenwert unmittelbar.

Wenn Sie den Wert eines veränderbaren Datentyps ändern (z. B. wie in dem vorstehenden Beispiel mit der Anweisung `del` und der Methode `append()`), werden die Werte unmittelbar ausgetauscht. Der Wert der Variablen wird aber nicht durch einen neuen Listenwert ersetzt.

Die Unterscheidung zwischen veränderbaren und unveränderbaren Typen mag wie sinnlose Haarspaltereи wirken, aber im Abschnitt »Verweise übergeben« weiter hinten in diesem Kapitel lernen Sie die unterschiedlichen Verhaltensweisen kennen, die sich je nachdem ergeben, ob Sie eine Funktion mit veränderbaren oder unveränderbaren Argumenten aufrufen. Bevor wir dorthin kommen, sehen wir uns jedoch noch den Datentyp der Tupel an, der eine unveränderbare Form des Listendatentyps darstellt.

## Der Datentyp für Tupel

Der Datentyp für *Tupel* ist fast identisch mit dem für Listen, wobei jedoch zwei wichtige Unterschiede bestehen. Erstens werden Tupel in runden statt in eckigen Klammern angegeben, wie das folgende Beispiel zeigt:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

Der Hauptunterschied zwischen Listen und Tupeln besteht jedoch darin, dass Tupel ebenso wie Strings unveränderbar sind. Es ist nicht möglich, ihre Werte zu bearbeiten, zu entfernen oder neue Werte anzuhängen. Der folgende Code führt daher zu der Fehlermeldung `TypeError`:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

Wenn sich in dem Tupel nur ein einziger Wert befindet, können Sie das dadurch kennzeichnen, dass Sie in den Klammern ein Komma hinter diesem Wert angeben. Andernfalls würde Python davon ausgehen, dass Sie einfach nur einen Wert in Klammern geschrieben haben. Anhand des Kommas kann Python erkennen, dass es sich um einen Tupelwert handelt. (Im Gegensatz zu anderen Programmiersprachen ist es in Python zulässig, ein Komma hinter das letzte Element in einer Liste oder einem Tupel zu setzen.) Um den Unterschied zu erkennen, geben Sie in der interaktiven Shell folgende `type()`-Funktionen ein:

```
>>> type(('hello',))
<class 'tuple'>
>>> type('hello')
<class 'str'>
```

Mithilfe von Tupeln machen Sie für die Leser des Codes deutlich, dass diese Folge von Werten nicht geändert werden soll. Wenn Sie also eine geordnete Folge von Werten brauchen, die sich nie ändern, dann verwenden Sie dafür ein Tupel. Der zweite Vorteil von Tupeln gegenüber Listen besteht darin, dass Python aufgrund

ihrer Unveränderbarkeit einige Optimierungen vornehmen kann, sodass der Code ein wenig schneller ausgeführt wird als bei der Verwendung von Listen.

### **Typen mit den Funktionen `list()` und `tuple()` umwandeln**

So wie `str(42)` die Stringdarstellung '42' des Integerwerts 42 zurückgibt, so geben die Funktionen `list()` und `tuple()` die Listen- bzw. Tupelversionen der übergebenen Werte zurück. In den folgenden Beispielen weisen die Rückgabewerte einen anderen Datentyp auf als die übergebenen Werte:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list('cat', 'dog', 5)
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

Die Umwandlung eines Tupels in eine Liste ist praktisch, wenn Sie eine veränderbare Version eines Tupelwerts benötigen.

## **Verweise**

Wie Sie bereits wissen, können Sie in Variablen Strings und Integerwerte speichern. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Hier weisen Sie der Variablen `spam` den Wert 42 zu. Anschließend kopieren Sie den Wert von `spam` und weisen ihn der Variablen `cheese` zu. Wenn Sie den Wert in `spam` später auf 100 ändern, hat das keinen Einfluss auf den Wert in `cheese`, denn schließlich sind `spam` und `cheese` voneinander unabhängige Variablen mit unterschiedlichen Werten.

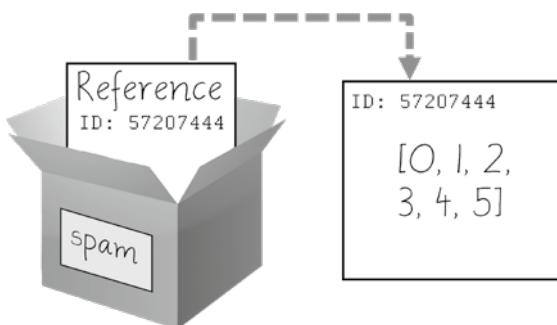
Listen aber funktionieren nicht auf dieselbe Weise. Wenn Sie einer Variablen eine Liste zuweisen, dann weisen Sie ihr in Wirklichkeit einen *Listenverweis* zu. Ein *Verweis* (oder eine *Referenz*) ist ein Wert, der auf irgendwelche Daten zeigt, und ein Listenverweis ist ein Wert, der auf eine Liste zeigt. Der folgende Code macht diese Unterscheidung deutlich:

```
>>> spam = [0, 1, 2, 3, 4, 5]    ❶
>>> cheese = spam    ❷
>>> cheese[1] = 'Hello!'    ❸
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```

Das mag Ihnen merkwürdig vorkommen. Schließlich hat der Code nur die Liste `cheese` bearbeitet, aber es sieht so aus, als wäre auch die Liste `spam` verändert worden!

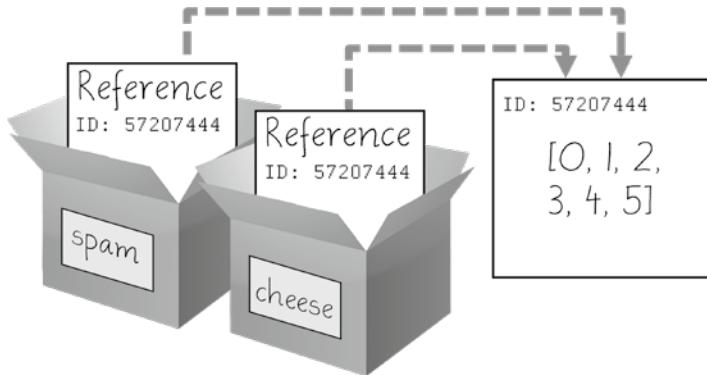
Wenn Sie die Liste erstellen (❶), weisen Sie der Variablen `spam` einen Verweis darauf zu. In der nächsten Zeile aber (❷) wird nicht der Listenwert selbst in `cheese` kopiert, sondern nur der Listenverweis, der sich in `spam` befindet. Das bedeutet, die in `spam` und `cheese` gespeicherten Werte verweisen nun beide auf dieselbe Liste. Es gibt nur eine einzige zugrunde liegende Liste, da die Liste selbst gar nicht kopiert wurde. Wenn Sie daher das erste Element von `cheese` ändern (❸), bearbeiten Sie in Wirklichkeit dieselbe Liste, auf die auch `spam` verweist.

Bekanntlich können Sie sich Variablen wie Kisten vorstellen, die Werte enthalten. Die bisherigen Darstellungen von Listen in Kisten in diesem Kapitel waren jedoch nicht ganz korrekt, denn in Wirklichkeit enthalten die Variablen gar nicht die Listen selbst, sondern nur *Verweise* auf Listen. (Diese Verweise tragen ID-Nummern, die Python intern verwendet, aber darum brauchen wir uns nicht zu kümmern.) Mit unserer Kistenmetapher für Variablen zeigt Abb. 4–4, was geschieht, wenn eine Liste der Variablen `spam` zugewiesen wird.



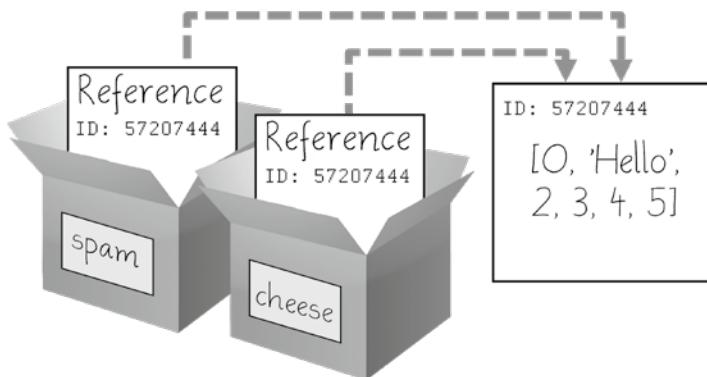
**Abb. 4–4** Mit `spam = [0, 1, 2, 3, 4, 5]` wird ein Verweis auf die Liste gespeichert, aber nicht die Liste selbst.

In Abb. 4–5 sehen Sie, wie der Verweis in `spam` zu `cheese` kopiert wird. Das bedeutet, dass nur ein neuer Verweis erstellt und in `cheese` gespeichert wird, aber keine neue Liste. Beide verweisen auf dieselbe Liste.



**Abb. 4–5** Mit `spam = cheese` wird der Verweis kopiert, aber nicht die Liste.

Wenn Sie die Liste ändern, auf die `cheese` verweist, ändern Sie damit auch die Liste, auf die `spam` verweist, da es sich dabei um ein und dieselbe Liste handelt. Das können Sie in Abb. 4–6 erkennen.



**Abb. 4–6** `cheese[1] = 'Hello!'` ändert die Liste, auf die beide Variablen verweisen.

Variablen enthalten Verweise auf Listenwerte und nicht die Listenwerte selbst. Strings und Integerwerte dagegen werden direkt in Variablen gespeichert. Python verwendet Verweise, wenn die Variablen Werte veränderbarer Datentypen speichern müssen, z. B. Listen oder Dictionarys, während die Werte unveränderbarer Datentypen wie Strings, Integer und Tupel direkt gespeichert werden.

Technisch gesehen enthalten Python-Variablen Referenzen auf Listen- oder Dictionary-Werte, doch umgangssprachlich wird oft die Formulierung gebraucht, dass eine Variable eine Liste oder ein Dictionary enthält.

## Verweise übergeben

Die Kenntnis von Verweisen ist unverzichtbar, um zu verstehen, wie Argumente an Funktionen übergeben werden. Bei einem Funktionsaufruf werden die Werte der Argumente in die Parameterlisten kopiert. Bei Listen (und bei Dictionarys, die ich im nächsten Kapitel beschreiben werde) heißt das, dass für den Parameter eine Kopie des Verweises verwendet wird. Um zu erkennen, was das bedeutet, geben Sie den folgenden Code im Dateieditor ein und speichern ihn als *passingReference.py*:

```
def eggs(someParameter):
    someParameter.append('Hello')

spam = [1, 2, 3]
eggs(spam)
print(spam)
```

Beim Aufruf von `eggs()` wird kein Rückgabewert verwendet, um `spam` einen neuen Wert zuzuweisen. Stattdessen wird die Liste unmittelbar geändert. Dieses Programm führt zu folgender Ausgabe:

```
[1, 2, 3, 'Hello']
```

Obwohl `spam` und `someParameter` getrennte Verweise enthalten, verweisen doch beide auf dieselbe Liste. Daher wirkt sich der Methodenaufruf `append('Hello')` innerhalb der Funktion auch dann noch auf die Liste aus, wenn der Funktionsaufruf die Steuerung zurückgegeben hat.

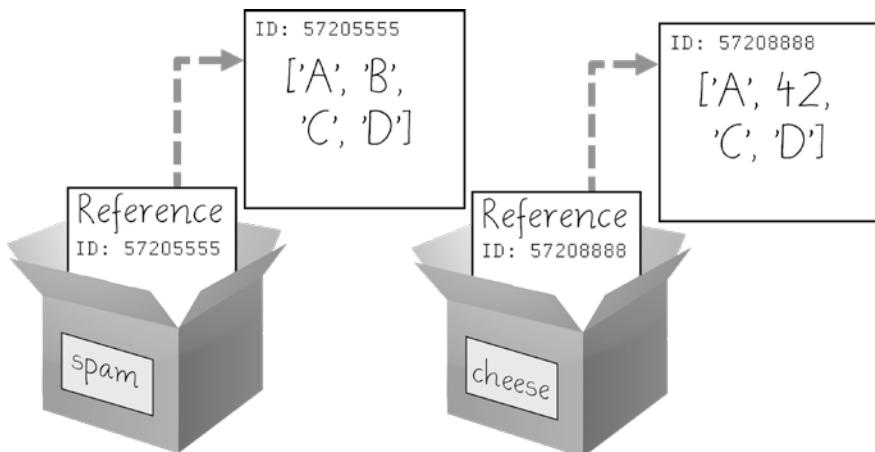
Merken Sie sich dieses Verhalten. Wenn Sie nicht beachten, dass Python Listen- und Dictionary-Variablen auf diese Weise handhabt, kann das zu verwirrenden Fehlern führen.

## Die Funktionen `copy()` und `deepcopy()` des Moduls `copy`

Die Übergabe von Verweisen ist oft die sinnvollste Vorgehensweise, um mit Listen und Dictionarys umzugehen, wenn die Funktion tatsächlich die übergebene Liste oder das Dictionary bearbeiten soll. Es kann jedoch auch vorkommen, dass Sie die ursprüngliche Liste bzw. das Dictionary nicht ändern wollen. Für diesen Fall stellt Python das Modul `copy` bereit, das die Funktionen `copy()` und `deepcopy()` enthält. Mit `copy()` können Sie eine echte Kopie eines veränderbaren Werts wie einer Liste oder eines Dictionarys erstellen, also nicht einfach nur eine Kopie des Verweises darauf. Geben Sie folgenden Code in die interaktive Shell ein:

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Hier verweisen die Variablen `spam` und `cheese` auf zwei unterschiedliche Listen, weshalb bei der Zuweisung von 42 zum Index 7 ausschließlich die Liste in `cheese` geändert wird. Wie Abb. 4–7 zeigt, verwenden die beiden Variablen nicht mehr dieselbe ID-Nummer, da sie nun auf verschiedene Listen verweisen.



**Abb. 4–7** `cheese = copy.copy(spam)` erstellt eine zweite Liste, die unabhängig von der ersten bearbeitet werden kann.

Wenn die Liste, die Sie kopieren wollen, selbst Listen enthält, verwenden Sie anstelle von `copy.copy()` die Funktion `copy.deepcopy()`. Damit werden auch die verschachtelten Listen kopiert.

## Zusammenfassung

Listen bilden einen sehr nützlichen Datentyp, da Sie damit Code schreiben können, der eine beliebige Anzahl von Werten in einer einzigen Variablen bearbeiten kann. Weiter hinten in diesem Buch werden Sie Programme kennenlernen, in denen Listen für Aufgaben eingesetzt werden, die ohne sie nur sehr schwer oder überhaupt nicht zu erledigen wären.

Listen sind veränderbar, was bedeutet, dass ihre Inhalte geändert werden können. Tupel und Strings dagegen ähneln zwar in gewisser Hinsicht Listen, sind aber unveränderbar. Eine Variable, die ein Tupel oder einen String enthält, kann mit einem neuen Tupel oder String überschrieben werden, aber das ist nicht das Gleiche wie die unmittelbare Veränderung eines Werts, wie sie beispielsweise mit den Methoden `append()` und `remove()` für Listen möglich ist.

In Variablen werden Listen nicht direkt gespeichert, sondern nur in Form von *Verweisen* auf die Listen. Dieser Unterschied ist von großer Bedeutung, wenn Variablen kopiert oder Listen als Argumente an Funktionsaufrufe übergeben werden. Da der kopierte Wert nur der Verweis auf die Liste ist, können sich jegliche Änderungen, die Sie an der Liste vornehmen, auch auf andere Variablen in dem Programm auswirken. Wenn Sie eine Liste in einer Variablen bearbeiten wollen, ohne dabei die ursprüngliche Liste zu ändern, müssen Sie sie mit `copy()` oder `deepcopy()` kopieren.

## Wiederholungsfragen

1. Was ist `[]`?
2. Wie weisen Sie den Wert `'hello'` als drittes Element der Liste zu, die in der Variablen `spam` gespeichert ist? (Nehmen Sie an, `spam` enthält `[2, 4, 6, 8, 10]`).

Nehmen Sie für die folgenden drei Fragen an, dass `spam` die Liste `['a', 'b', 'c', 'd']` enthält.

3. Wozu wird `spam[int(int('3' * 2) / 11)]` ausgewertet?
4. Wozu wird `spam[-1]` ausgewertet?
5. Wozu wird `spam[:2]` ausgewertet?

Nehmen Sie für die folgenden drei Fragen an, dass `bacon` die Liste `[3.14, 'cat', 11, 'cat', True]` enthält.

6. Wozu wird `bacon.index('cat')` ausgewertet?
7. Wie sieht der Listenwert in `bacon` nach der Ausführung von `bacon.append(99)` aus?
8. Wie sieht der Listenwert in `bacon` nach der Ausführung von `bacon.remove('cat')` aus?
9. Wie sehen die Operatoren für die Listenverkettung und die Listenwiederholung aus?
10. Was ist der Unterschied zwischen den Listenmethoden `append()` und `insert()`?
11. Welche beiden Möglichkeiten gibt es, um Elemente aus einer Liste zu entfernen?
12. Nennen Sie einige Ähnlichkeiten zwischen Listen- und Stringwerten!

13. Worin unterscheiden sich Listen und Tupel?
14. Wie geben Sie einen Tupelwert an, der nur den Integerwert 42 enthält?
15. Wie können Sie die Tupelform eines Listenwerts erzeugen? Wie die Listenform eines Tupelwerts?
16. In Variablen, die Listenwerte »enthalten«, ist die eigentliche Liste gar nicht gespeichert. Was enthalten sie in Wirklichkeit?
17. Was ist der Unterschied zwischen `copy.copy()` und `copy.deepcopy()`?

## Übungsprojekte

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erledigen:

### Kommacode

Nehmen Sie an, Sie haben einen Listenwert wie den folgenden:

```
spam = ['apples', 'bananas', 'tofu', 'cats']
```

Schreiben Sie eine Funktion, die einen Listenwert als Argument annimmt und einen String zurückgibt, in dem alle Listenelemente durch ein Komma und ein Leerzeichen getrennt sind und vor dem letzten Eintrag *and* steht. Wenn Sie beispielsweise die vorstehende Liste `spam` an diese Funktion übergeben, sollte sie '`apples, bananas, tofu, and cats`' zurückgeben. Die Funktion muss allerdings bei jedem beliebigen Listenwert funktionieren.

### Zeichenbildraster

Gegeben sei eine Liste von Listen, bei der jedes Element der inneren Listen ein String aus einem Zeichen ist:

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', '0', '0', '.', '.', '.'],
        ['0', '0', '0', '0', '.', '.'],
        ['0', '0', '0', '0', '0', '.'],
        ['.', '0', '0', '0', '0', '0'],
        ['0', '0', '0', '0', '0', '.'],
        ['0', '0', '0', '0', '.', '.'],
        ['.', '0', '0', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

Sie können sich `grid[x][y]` als das Zeichen an der x- und y-Koordinate eines Bilds vorstellen, das mithilfe von Textzeichen dargestellt wird. Der Ursprung (0, 0) liegt dabei in der oberen linken Ecke, die Koordinate x wird nach rechts gezählt, die Koordinate y nach unten.

Kopieren Sie den vorstehenden Rasterwert und schreiben Sie Code, der damit das folgende Bild ausgibt:

```
..00.00..  
.0000000.  
.0000000.  
..00000..  
...000...  
....0....
```

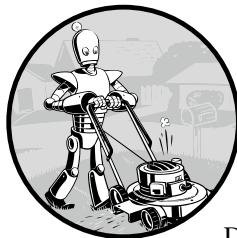
Tipp: Sie brauchen eine Schleife innerhalb einer Schleife. Damit geben Sie zunächst `grid[0][0]`, `grid[1][0]`, `grid[2][0]` usw. bis `grid[8][0]` aus. Damit ist die erste Zeile vollständig, sodass Sie danach einen Zeilenumbruch einschalten. Anschließend muss das Programm `grid[0][1]`, `grid[1][1]`, `grid[2][1]` usw. ausgeben. Als Letztes wird `grid[8][5]` ausgegeben.

Denken Sie daran, das Schlüsselwortargument `end` an `print()` zu übergeben, um zu verhindern, dass nach jedem Aufruf von `print()` automatisch ein Zeilenumbruch erfolgt.



# 5

## Dictionarys und Datenstrukturen



In diesem Kapitel geht es um den Datentyp der Dictionarys (Wörterbücher), der eine flexible Möglichkeit bietet, um Daten zu gliedern und darauf zuzugreifen. Durch die Kombination von Dictionarys mit den aus dem letzten Kapitel bekannten Listen sind Sie in der Lage, Datenstrukturen zu erstellen, um beispielsweise ein Tic-Tac-Toe-Spielfeld zu modellieren.

### Der Datentyp für Dictionarys

Ebenso wie Listen sind *Dictionarys* Zusammenstellungen mehrerer Werte. Die Indizes in Dictionarys können aber nicht nur Integer sein wie bei Listen, sondern auch andere Datentypen aufweisen. Sie werden *Schlüssel* genannt und die Kombination eines Schlüssels mit dem zugehörigen Wert heißt *Schlüssel-Wert-Paar*.

Dictionarys werden mithilfe von geschweiften Klammern geschrieben:

```
>>> myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

Mit dieser Codezeile wird der Variable `myCat` ein Dictionary zugewiesen. Die Schlüssel in diesem Dictionary sind `'size'`, `'color'` und `'disposition'` mit den Werten `'fat'`, `'gray'` bzw. `'loud'`. Mithilfe der Schlüssel können Sie die Werte abrufen:

```
>>> myCat['size']
'fat'
>>> 'My cat has ' + myCat['color'] + ' fur.'
'My cat has gray fur.'
```

In Dictionarys können auch Integer als Schlüssel verwendet werden. Allerdings müssen sie im Gegensatz zu den Indizes von Listen nicht bei null beginnen, sondern können ganz beliebig sein:

```
>>> spam = {12345: 'Luggage Combination', 42: 'The Answer'}
```

## Dictionaries und Listen im Vergleich

Im Gegensatz zu Listen sind Dictionarys nicht geordnet. Das erste Element in der Liste `spam` ist `spam[0]`, aber in einem Dictionary gibt es kein erstes Element. Bei der Frage, ob zwei Listen gleich sind, spielt die Reihenfolge der Elemente eine Rolle, aber bei einem Dictionary ist es ohne Bedeutung, in welcher Reihenfolge die Schlüssel-Wert-Paare stehen. Geben Sie zur Veranschaulichung Folgendes in die interaktive Shell ein:

```
>>> spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
True
```

Da Dictionarys ungeordnet sind, können Sie auch keine Slices daraus entnehmen. Wenn Sie versuchen, auf einen Schlüssel zuzugreifen, den es in dem Dictionary nicht gibt, erhalten Sie die Fehlermeldung `KeyError` – vergleichbar mit dem `IndexError`, der sich ergibt, wenn Sie bei einer Liste einen nicht existierenden Index abfragen. Im folgenden Beispiel erscheint diese Fehlermeldung, da es den Schlüssel `'color'` nicht gibt:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    spam['color']
KeyError: 'color'
```

Dictionarys sind zwar ungeordnet, aber da Sie darin willkürliche Werte als Schlüssel verwenden können, stellen sie eine äußerst vielseitige und nützliche Möglichkeit dar, um Daten zu gliedern. Nehmen wir an, Sie möchten in einem Programm die Geburtsdaten Ihrer Freunde speichern. Dazu können Sie ein Dictionary verwenden, bei dem die Namen dieser Personen die Schlüssel und die Geburtsdaten die Werte sind. Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *birthdays.py*:

```
birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4'} ❶

while True:
    print('Enter a name: (blank to quit)')
    name = input()
    if name == '':
        break

    if name in birthdays: ❷
        print(birthdays[name] + ' is the birthday of ' + name) ❸
    else:
        print('I do not have birthday information for ' + name)
        print('What is their birthday?')
        bday = input()
        birthdays[name] = bday ❹
        print('Birthday database updated.')
```

Hier erstellen Sie zunächst ein Grund-Dictionary und speichern es in *birthdays* (❶). Ob ein Name bereits in dem Dictionary vorhanden ist, können Sie (ebenso wie bei Listen) mit dem Schlüsselwort `in` herausfinden (❷). Wenn der Name schon gespeichert ist, können Sie den zugehörigen Wert mithilfe von eckigen Klammern abrufen (❸). Wenn nicht, verwenden Sie die gleiche Schreibweise mit eckigen Klammern und den Zuweisungsoperator, um den Namen und einen Wert hinzuzufügen (❹).

Die Ausgabe dieses Programms sieht wie folgt aus:

```
Enter a name: (blank to quit)
Alice
Apr 1 is the birthday of Alice
Enter a name: (blank to quit)
Eve
I do not have birthday information for Eve
What is their birthday?
Dec 5
Birthday database updated.
Enter a name: (blank to quit)
Eve
Dec 5 is the birthday of Eve
Enter a name: (blank to quit)
```

Am Ende des Programms werden jedoch alle eingegebenen Daten verworfen. Wie Sie Daten in Dateien auf Ihrer Festplatte speichern, erfahren Sie in Kapitel 8.

### Die Methoden `keys()`, `values()` und `items()`

Dictionarys verfügen über drei Methoden, die listenähnliche Werte mit den Schlüsseln, den Werten bzw. den Schlüssel-Wert-Paaren zurückgeben, nämlich `keys()`, `values()` und `items()`. Die Rückgabewerte sind jedoch keine echten Listen, denn sie können nicht verändert werden und haben auch keine `append()`-Methode. Die Datentypen dieser Werte (`dict_keys`, `dict_values` und `dict_items`) können jedoch in `for`-Schleifen verwendet werden. Um zu sehen, wie das funktioniert, geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
    print(v)

red
42
```

Hier durchläuft die `for`-Schleife alle Werte im Dictionary `spam`. Es ist jedoch auch eine Iteration über alle Schlüssel oder alle Schlüssel-Wert-Paare möglich:

```
>>> for k in spam.keys():
    print(k)

color
age
>>> for i in spam.items():
    print(i)

('color', 'red')
('age', 42)
```

Unter Verwendung der Methoden `keys()`, `values()` und `items()` kann eine `for`-Schleife also über die Schlüssel, die Werte bzw. die Schlüssel-Wert-Paare eines Dictionarys iterieren. Beachten Sie, dass die von der Methode `items()` zurückgegebenen `dict_items`-Werte Tupel aus Schlüssel und Wert sind.

Wenn Sie aus diesen Methoden echte Listen gewinnen möchten, übergeben Sie die listenähnlichen Rückgabewerte an die Funktion `list()`:

```
>>> spam = {'color': 'red', 'age': 42}
>>> spam.keys()
dict_keys(['color', 'age'])
>>> list(spam.keys())
['color', 'age']
```

In der Zeile `list(spam.keys())` wird der von `keys()` zurückgegebene `dict_keys`-Wert an die Funktion `list()` übergeben, die den Listenwert `['color', 'age']` zurückgibt.

In der `for`-Schleife können Sie auch eine Mehrfachzuweisung durchführen, um Schlüssel und Wert unterschiedlichen Variablen zuzuweisen:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
    print('Key: ' + k + ' Value: ' + str(v))

Key: age Value: 42
Key: color Value: red
```

## Das Vorhandensein eines Schlüssels oder Werts im Dictionary ermitteln

Im vorigen Kapitel haben Sie gelernt, dass Sie mit den Operatoren `in` und `not in` prüfen können, ob ein gegebener Wert in einer Liste vorhanden ist. Mithilfe dieser Operatoren lässt sich jedoch auch ermitteln, ob ein bestimmter Schlüssel oder Wert in einem Dictionary existiert:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
>>> 'color' not in spam.keys()
True
>>> 'color' in spam
False
```

Die Schreibweise `'color' in spam` in diesem Beispiel ist im Grunde genommen eine Abkürzung für `'color' in spam.keys()`. Das können Sie immer so machen: Wenn Sie prüfen wollen, ob ein Schlüssel in einem Dictionary vorkommt (oder nicht), wenden Sie das Schlüsselwort `in` (bzw. `not in`) einfach auf den Dictionary-Wert an.

## Die Methode `get()`

Es wäre ziemlich mühselig, wenn man beim Abruf des Werts für einen Schlüssel in einem Dictionary zunächst immer prüfen müsste, ob dieser Schlüssel überhaupt vorhanden ist. Zum Glück gibt es für Dictionarys die Methode `get()`, die zwei Argumente annimmt, nämlich den Schlüssel des abzurufenden Werts und einen Standardwert, der zurückgegeben werden soll, wenn es den Schlüssel nicht gibt.

Geben Sie Folgendes in die interaktive Shell ein:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

Da im Dictionary `picnicItems` kein Schlüssel namens 'eggs' vorhanden ist, gibt die Methode `get()` den Standardwert 0 zurück. Hätten Sie die Abfrage ohne die Methode `get()` versucht, so wäre eine Fehlermeldung die Folge gewesen:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs'
```

### Die Methode `setdefault()`

Es kommt häufig vor, dass Sie in einem Dictionary nach einem Schlüssel suchen und ihm nur dann einen Wert hinzufügen wollen, wenn er noch keinen hat. Das können Sie mit folgendem Code erreichen:

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
    spam['color'] = 'black'
```

Mithilfe der Methode `setdefault()` können Sie das in einer einzigen Codezeile erledigen. Das erste Argument dieser Methode ist der Schlüssel, nach dem Sie suchen, das zweite der Wert, der dafür festgelegt werden soll, falls der Schlüssel noch nicht existiert. Ist der Schlüssel schon vorhanden, so gibt `setdefault()` dessen bestehenden Wert zurück. Das können Sie sich ansehen, indem Sie folgenden Code in die interaktive Shell eingeben:

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

Beim ersten Aufruf von `setdefault()` wird das Dictionary in `spam` in `{'color': 'black', 'age': 5, 'name': 'Pooka'}` geändert. Die Methode gibt `'black'` zurück, da dies der Wert ist, der für den Schlüssel `'color'` festgelegt wurde. Wird danach `spam.setdefault('color', 'white')` aufgerufen, wird der Wert des Schlüssels `'color'` jedoch *nicht* in `'white'` geöffnet, da dieser Schlüssel bereits vorhanden ist.

Die Methode `setdefault()` stellt eine praktische Abkürzung dar, um sicherzustellen, dass ein Schlüssel vorhanden ist. Das folgende kurze Programm zählt, wie oft die einzelnen Zeichen in einem String vorkommen. Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *characterCount.py*:

```
message = 'It was a bright cold day in April, and the clocks were striking  
thirteen.'  
count = {}  
  
for character in message:  
    count.setdefault(character, 0)  
    count[character] = count[character] + 1  
  
print(count)
```

Das Programm durchläuft alle Zeichen im String der Variable `message` und zählt, wie oft sie jeweils vorkommen. Der Aufruf der Methode `setdefault()` sorgt dafür, dass der Schlüssel `character` (mit dem Standardwert 0) im Dictionary `count` vorhanden ist, damit das Programm bei der Ausführung von `count[character] = count[character] + 1` keinen `KeyError` hervorruft. Die Ausgabe dieses Programms sieht wie folgt aus:

```
{ ' ': 13, ',': 1, '.': 1, 'A': 1, 'I': 1, 'a': 4, 'c': 3, 'b': 1, 'e': 5,
'd': 3, 'g': 2, 'i': 6, 'h': 3, 'k': 2, 'l': 3, 'o': 2, 'n': 4, 'p': 1,
's': 3, 'r': 5, 't': 6, 'w': 2, 'y': 1}
```

Hieraus können Sie ablesen, dass das kleine *c* dreimal vorkommt, das Leerzeichen 13 Mal und das große *A* einmal. Dieses Programm funktioniert unabhängig davon, wie der String in der Variable `message` aussieht, selbst wenn einer eine Million Zeichen lang ist!

## **Saubere Ausgabe**

Wenn Sie das Modul `pprint` importieren, haben Sie Zugriff auf die Funktionen `pprint()` und `pformat()`, mit denen Sie ein Dictionary sauber ausgeben können (»pretty print«). Das ist hilfreich, wenn Sie eine übersichtlichere Anzeige der Elemente in einem Dictionary brauchen, als `print()` sie bietet. Ändern Sie das Programm `characterCount.py` wie folgt ab und speichern Sie es als `prettyCharacterCount.py`:

```
import pprint
message = 'It was a bright cold day in April, and the clocks were striking
          thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

pprint.pprint(count)
```

Die Ausgabe dieses Programm ist nach Schlüsseln geordnet und viel übersichtlicher:

```
{' ': 13,
',': 1,
'.': 1,
'A': 1,
'I': 1,
'a': 4,
'b': 1,
'c': 3,
'd': 3,
'e': 5,
'g': 2,
'h': 3,
'i': 6,
'k': 2,
'l': 3,
'n': 4,
'o': 2,
'p': 1,
'r': 5,
's': 3,
't': 6,
'w': 2,
'y': 1}
```

Die Funktion `pprint.pprint()` ist insbesondere dann hilfreich, wenn das Dictionary selbst wiederum Listen oder andere Dictionaries als Werte enthält.

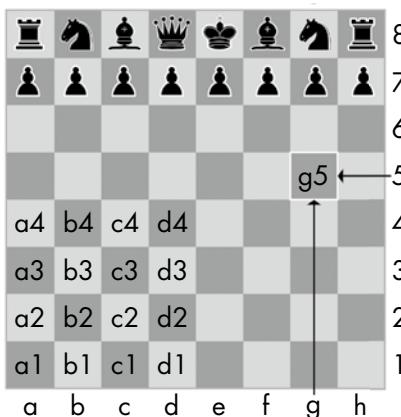
Wenn Sie den aufgehübschten Text nicht auf dem Bildschirm ausgeben, sondern in einen Stringwert umwandeln möchten, rufen Sie stattdessen `pprint.format()` auf. Die folgenden beiden Codezeilen sind gleichwertig:

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

## Datenstrukturen zur Modellierung realer Objekte

Schon bevor es das Internet gab, war es möglich, mit einer Person zu spielen, die sich am anderen Ende der Welt befand. Beide Spieler stellten dabei jeweils ein Schachbrett bei sich zu Hause auf und sandten sich abwechselnd Postkarten mit ihren Spielzügen zu. Dazu brauchten die Spieler natürlich eine eindeutige Möglichkeit, um die Züge zu beschreiben.

In der *algebraischen Schachnotation* werden die Felder auf dem Schachbrett durch Koordinaten aus Zahlen und Buchstaben bezeichnet, wie Sie in Abb. 5–1 sehen.



**Abb. 5–1** Die Koordinaten auf einem Schachbrett in algebraischer Schachnotation

Auch die Figuren werden durch Buchstaben bezeichnet: *K* für König, *D* für Dame, *T* für Turm, *S* für Springer, *L* für Läufer. Die Beschreibung des Zugs einer Partei (Halbzug) besteht aus dem Buchstaben für die Figur und den Zielkoordinaten. Ein Paar solcher Halbzüge gibt an, was bei einem kompletten Zug geschieht (wobei Weiß vorn steht). So bedeutet *2. Sf3 Sc6*, dass Weiß im zweiten Zug einen seiner Springer auf Feld *f3* gezogen hat und Schwarz einen Springer auf Feld *c6*.

Damit ist die algebraische Notation natürlich noch nicht komplett beschrieben. Wichtig für unsere Zwecke ist jedoch, dass es damit möglich ist, eine Schachpartie eindeutig zu beschreiben, ohne dass die beiden Spieler am selben Brett sitzen müssen; sie können sich durchaus auf verschiedenen Kontinenten befinden. Wenn sie ein gutes Gedächtnis haben, brauchen sie nicht einmal ein echtes Schachbrett, sondern können einfach den Spielzug auf der Postkarte lesen und den Spielstand im Kopf aktualisieren.

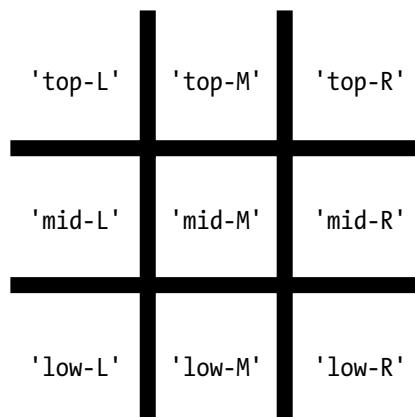
Computer haben ein gutes Gedächtnis. Ein Programm auf einem modernen Rechner kann problemlos Milliarden von Zügen wie '*2. Sf3 Sc6*' speichern.

Dadurch können Computer Schach spielen, ohne auf ein physisches Schachbrett zurückzugreifen. Sie modellieren Daten, um ein Schachbrett darzustellen. Der Code dient dann dazu, mit diesem Modell zu arbeiten.

Dabei kommen Listen und Dictionaries ins Spiel, denn mit ihrer Hilfe können Sie reale Objekte modellieren, beispielsweise Schachbretter. Für unser erstes Beispiel wählen wir jedoch ein einfacheres Spiel als Schach: Tic-Tac-Toe.

### Ein Tic-Tac-Toe-Brett

Ein Tic-Tac-Toe-Brett sieht wie ein überdimensioniertes Doppelkreuz (#) mit neun Feldern aus, die jeweils ein X, ein 0 oder gar nichts enthalten können. Um ein solches Brett durch ein Dictionary darzustellen, weisen Sie jedem Feld einen Schlüssel in Form eines Strings zu, wie Sie in Abb. 5–2 sehen.



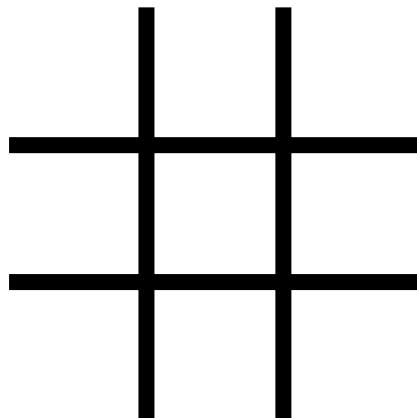
**Abb. 5–2** Die Felder eines Tic-Tac-Toe-Bretts mit den zugehörigen Schlüsseln

Was sich auf den einzelnen Feldern des Bretts befindet, können Sie ebenfalls durch Stringwerte darstellen: 'X', '0' oder ' ' (ein Leerzeichen für ein leeres Feld). Dabei müssen Sie neun dieser Stringwerte speichern. Dazu verwenden Sie ein Dictionary. Der Stringwert mit dem Schlüssel 'top-R' steht für den Inhalt des Felds in der rechten oberen Ecke, der Stringwert mit dem Schlüssel 'low-L' für das Feld unten links, der Stringwert mit dem Schlüssel 'mid-M' für das mittlere Feld usw.

Dieses Dictionary ist eine Datenstruktur, die ein Tic-Tac-Toe-Brett darstellt. Speichern Sie es in der Variablen `theBoard`. Geben Sie dazu den folgenden Quellcode im Dateieditor ein und speichern Sie ihn als `ticTacToe.py`:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Die Datenstruktur in der Variablen `theBoard` steht für das Tic-Tac-Toe-Brett aus Abb. 5–3.

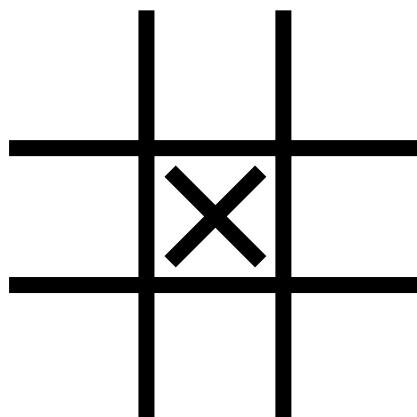


**Abb. 5–3** Ein leeres Tic-Tac-Toe-Brett

Da die Werte zu den Schlüsseln in `theBoard` jeweils Strings aus einem Zeichen sind, stellt dieses Dictionary ein komplett leeres Brett dar. Wenn der X-Spieler anfängt und sein Kreuz im mittleren Feld macht, wird das Brett in diesem Zustand durch folgendes Dictionary wiedergegeben:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',  
           'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',  
           'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Die Datenstruktur in `theBoard` stellt jetzt das Tic-Tac-Toe-Brett aus Abb. 5–4 dar.

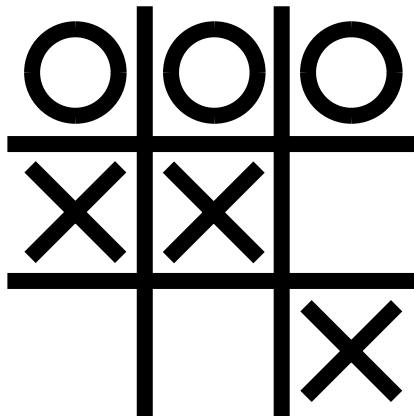


**Abb. 5–4** Der erste Zug

Wenn der O-Spieler gewinnt, nachdem er die obere Reihe komplett ausgefüllt hat, sieht das Brett wie folgt aus:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
            'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

Diese Datenstruktur steht für das Brett mit dem Spielstand aus Abb. 5–5.



**Abb. 5–5** Spieler O gewinnt.

Die Spieler können den Inhalt der Variablen jedoch nicht sehen, sondern nur das, was auf dem Bildschirm ausgegeben wird. Daher schreiben wir eine Funktion, die das Dictionary für das Brett anzeigt. Ergänzen Sie *ticTacToe.py* wie folgt (der neu hinzugekommene Code ist fett dargestellt):

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
def printBoard(board):
    print(board['top-L'] + ' | ' + board['top-M'] + ' | ' + board['top-R'])
    print('---+---')
    print(board['mid-L'] + ' | ' + board['mid-M'] + ' | ' + board['mid-R'])
    print('---+---')
    print(board['low-L'] + ' | ' + board['low-M'] + ' | ' + board['low-R'])
printBoard(theBoard)
```

Wenn Sie dieses Programm ausführen, gibt `printBoard()` das leere Tic-Tac-Toe-Brett aus:

```

| |
-+--+
| |
-+--+
| |

```

Die Funktion `printBoard()` kann mit jeder Tic-Tac-Toe-Datenstruktur umgehen, die Sie ihr übergeben. Versuchen Sie, den Code wie folgt abzuändern:

```

theBoard = {'top-L': '0', 'top-M': '0', 'top-R': '0', 'mid-L': 'X', 'mid-M': 'X',
            'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}

def printBoard(board):
    print(board['top-L'] + ' | ' + board['top-M'] + ' | ' + board['top-R'])
    print(' -+-- ')
    print(board['mid-L'] + ' | ' + board['mid-M'] + ' | ' + board['mid-R'])
    print(' -+-- ')
    print(board['low-L'] + ' | ' + board['low-M'] + ' | ' + board['low-R'])
printBoard(theBoard)

```

Jetzt sieht die Ausgabe des Spielbretts auf dem Bildschirm wie folgt aus:

```

0|0|0
-+--+
X|X|
-+--+
| |X

```

Sie haben eine Datenstruktur erstellt, die das Tic-Tac-Toe-Brett wiedergibt, und in `printBoard()` Code geschrieben, der diese Struktur interpretiert. Damit haben Sie ein Programm, das das Spielbrett modelliert. Die Datenstruktur hätten Sie auch anderes gestalten können (z. B. durch Schlüssel wie 'TOP-LEFT' statt 'top-L'), aber solange der Code mit der Datenstruktur umgehen kann, funktioniert das Programm richtig.

Beispielsweise erwartet die Funktion `printBoard()`, dass die Tic-Tac-Toe-Datenstruktur ein Dictionary mit Schlüsseln für alle neun Felder ist. Wenn dem Dictionary, das dieser Funktion übergeben wird, ein Schlüssel fehlt, beispielsweise 'mid-L', funktioniert das Programm nicht mehr:

```

0|0|0
-+--+
Traceback (most recent call last):
  File "ticTacToe.py", line 10, in <module>
    printBoard(theBoard)
  File "ticTacToe.py", line 6, in printBoard
    print(board['mid-L'] + ' | ' + board['mid-M'] + ' | ' + board['mid-R'])
KeyError: 'mid-L'

```

Als Nächstes fügen wir Code hinzu, mit dem die Spieler ihre Züge eingeben können. Ändern Sie das Programm *ticTacToe.py* wie folgt ab:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ', 'mid-M':
' ', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

def printBoard(board):
    print(board['top-L'] + ' | ' + board['top-M'] + ' | ' + board['top-R'])
    print('---+---')
    print(board['mid-L'] + ' | ' + board['mid-M'] + ' | ' + board['mid-R'])
    print('---+---')
    print(board['low-L'] + ' | ' + board['low-M'] + ' | ' + board['low-R'])

turn = 'X'
for i in range(9):
    printBoard(theBoard) ❶
    print('Turn for ' + turn + '. Move on which space?')
    move = input() ❷
    theBoard[move] = turn ❸
    if turn == 'X': ❹
        turn = 'O'
    else:
        turn = 'X'
printBoard(theBoard)
```

Der neu hinzugefügte Code gibt zu Beginn jedes Zugs das Brett aus (❶), ruft dann den Zug des Spielers ab, der gerade an der Reihe ist (❷), aktualisiert daraufhin das Spielbrett (❸) und wechselt zum nächsten Spieler (❹), bevor der nächste Zug erfolgt.

Die Ausgabe sieht wie folgt aus:

```
| |
-+-+-
| |
-+-+-
| |
Turn for X. Move on which space?
mid-M
| |
-+-+-
|X|
-+-+-
| |
Turn for O. Move on which space?
low-L
| |
-+-+-
|X|
-+-+-
0| |
```

-- schnipp --

```
0|0|X
-+-+-
X|X|0
-+-+-
0| |X
Turn for X. Move on which space?
1ow-M
0|0|X
-+-+-
X|X|0
-+-+-
0|X|X
```

Dies ist natürlich noch kein komplettes Programm für ein Tic-Tac-Toe-Spiel, denn unter anderem wird nicht geprüft, welcher Spieler gewonnen hat. Es reicht aber aus, um Ihnen zu zeigen, wie Sie Datenstrukturen in Ihren Programmen einsetzen können.

### Hinweis

Der Quellcode für ein vollständiges Tic-Tac-Toe-Programm wird in den Ressourcen beschrieben, die Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) finden.

## Verschachtelte Dictionarys und Listen

Die Modellierung eines Tic-Tac-Toe-Bretts ist ziemlich einfach, denn dazu brauchen Sie nur ein einziges Dictionary mit neun Schlüssel-Wert-Paaren. Wenn Sie kompliziertere Dinge modellieren wollen, kann es jedoch sein, dass Sie dazu Dictionarys und Listen benötigen, die andere Dictionarys und Listen enthalten. Listen sind geeignet, wenn Sie eine geordnete Reihe von Werten brauchen, und Dictionaries, wenn Sie Werte mit Schlüsseln verknüpfen möchten. In dem folgenden Programm wird ein Dictionary verwendet, das selbst wiederum Dictionarys enthält, um festzuhalten, wer was zu einem Picknick mitbringt. Die Funktion `totalBrought()` kann diese Datenstruktur lesen und berechnen, wie in welcher Menge ein Nahrungsmittel oder Gegenstand insgesamt von allen Gästen mitgebracht wird.

```

allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
            'Bob': {'ham sandwiches': 3, 'apples': 2},
            'Carol': {'cups': 3, 'apple pies': 1}}


def totalBrought(guests, item):
    numBrought = 0
    for k, v in guests.items(): ❶
        numBrought = numBrought + v.get(item, 0) ❷
    return numBrought


print('Number of things being brought:')
print(' - Apples      ' + str(totalBrought(allGuests, 'apples')))
print(' - Cups        ' + str(totalBrought(allGuests, 'cups')))
print(' - Cakes       ' + str(totalBrought(allGuests, 'cakes')))
print(' - Ham Sandwiches ' + str(totalBrought(allGuests, 'ham sandwiches')))
print(' - Apple Pies   ' + str(totalBrought(allGuests, 'apple pies')))
```

In der Funktion `totalBrought()` durchläuft die `for`-Schleife die Schlüssel-Wert-Paare von `guests` (❶). In der Schleife wird der String für den Namen des Gasts zu `k` zugewiesen und das Dictionary der Dinge, die er mitbringt, zu `v`. Ist der Parameter eines Elements schon als Schlüssel in dem Dictionary vorhanden, wird sein Wert (die Menge) zu `numBrought` addiert (❷). Gibt es den Schlüssel noch nicht, gibt die Methode `get()` den Wert 0 zurück, der dann zu `numBrought` addiert wird.

Die Ausgabe dieses Programms sieht wie folgt aus:

```

Number of things being brought:
- Apples 7
- Cups 3
- Cakes 0
- Ham Sandwiches 3
- Apple Pies 1
```

Die Modellierung scheint so einfach zu sein, dass Sie sich vielleicht gar nicht erst die Mühe machen wollen, ein Programm dafür zu schreiben. Denken Sie aber daran, dass die Funktion `totalBrought()` auch problemlos Dictionaries mit Tausenden von Gästen handhaben kann, die jeder wiederum Tausende von Dingen zum Picknick mitbringen. Wenn Sie all diese Informationen in einer Datenstruktur haben und mit der Funktion `totalBrought()` verarbeiten können, sparen Sie eine Menge Zeit!

Mit Datenstrukturen können Sie reale Objekte auf beliebige Weise modellieren, solange der Rest des Programms nur korrekt mit diesem Datenmodell umgehen kann. Wenn Sie mit dem Programmieren anfangen, sollten Sie sich zunächst nicht allzu viele Gedanken darüber machen, was die »richtige« Art und Weise der Modellierung ist. Je mehr Erfahrung Sie gewinnen, umso effizientere Modelle werden Sie erstellen. Das Wichtigste ist, dass das Datenmodell für die Bedürfnisse des Programms geeignet ist.

## Zusammenfassung

In diesem Kapitel haben Sie Dictionarys kennengelernt. Listen und Dictionarys sind Werte, die mehrere andere Werte enthalten, bei denen es sich auch wiederum um Listen und Dictionarys handeln kann. Der Clou bei Dictionarys besteht darin, dass Sie damit einen Eintrag (den Schlüssel) einem anderen (dem Wert) zuordnen können, wohingegen Listen lediglich eine geordnete Folge von Werten enthalten. Um auf die Werte innerhalb eines Dictionarys zuzugreifen, verwenden Sie ebenso wie bei Listen die Schreibweise mit eckigen Klammern. Anstelle eines Integerindex werden bei Dictionarys Schlüssel genutzt, die verschiedene Datentypen aufweisen können – Integer, Fließkommazahlen, Strings und Tupel. Durch die Gliederung von Werten in Datenstrukturen können Sie reale Objekte modellieren. Das haben Sie am Beispiel des Tic-Tac-Toe-Bretts gesehen.

Damit haben wir die Grundlagen der Python-Programmierung auch schon abgedeckt. Im restlichen Verlauf dieses Buchs werden Sie noch weitere Begriffe und Prinzipien kennenlernen, aber Sie haben bereits jetzt ausreichend Kenntnisse, um nützliche Programme zur Automatisierung von Aufgaben zu schreiben. Sie mögen vielleicht einwenden, dass Sie noch nicht genug über Python wissen, um Webseiten herunterzuladen, Arbeitsblätter zu ändern oder Textnachrichten zu senden. Dafür aber gibt es Python-Module, die von anderen Programmierern geschrieben wurden und Funktionen enthalten, die solche Aufgaben erleichtern. Sehen wir uns nun also an, wie Sie echte Programme schreiben, die automatisch nützliche Dinge tun.

## Wiederholungsfragen

1. Wie sieht der Code für ein leeres Dictionary aus?
2. Wie sieht ein Dictionary mit dem Schlüssel 'foo' und dem Wert 42 aus?
3. Was ist der Hauptunterschied zwischen einem Dictionary und einer Liste?
4. Was geschieht, wenn Sie versuchen, `spam['foo']` abzurufen und `spam` den Wert `{'bar': 100}` hat?
5. Was ist der Unterschied zwischen `'cat'` in `spam` und `'cat'` in `spam.keys()`, wenn in `spam` ein Dictionary gespeichert ist?
6. Was ist der Unterschied zwischen `'cat'` in `spam` und `'cat'` in `spam.values()`, wenn in `spam` ein Dictionary gespeichert ist?
7. Wie können Sie den folgenden Code kürzer fassen?

```
if 'color' not in spam:  
    spam['color'] = 'black'
```

8. Welche Module und Funktionen können Sie verwenden, um Dictionarys auf saubere und übersichtliche Weise auszugeben?

## Übungsprojekte

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erledigen:

### Inventar für ein Fantasyspiel

Stellen Sie sich vor, Sie schreiben ein Fantasy-Computerspiel. Als Datenstruktur für das Inventar des Spielers verwenden Sie ein Dictionary. Die Schlüssel sind dabei Strings, die die einzelnen Posten in dem Inventar beschreiben, während der Wert ein Integer ist, der angibt, wie viele dieser Gegenstände der Spieler hat. Beispielsweise bedeutet der Dictionary-Wert `{'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}`, dass der Spieler über ein Seil, sechs Fackeln, 42 Goldmünzen, einen Dolch und zwölf Pfeile verfügt.

Schreiben Sie nun eine Funktion namens `displayInventory()`, die ein beliebiges Inventar entgegennimmt und wie folgt anzeigt:

```
Inventory:  
12 arrow  
42 gold coin  
1 rope  
6 torch  
1 dagger  
Total number of items: 62
```

Tipp: Um die Schlüssel in einem Dictionary zu durchlaufen, können Sie eine for-Schleife verwenden.

```
# inventory.py  
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}  
  
def displayInventory(inventory):  
    print("Inventory:")  
    item_total = 0  
    for k, v in inventory.items():  
        print(str(v) + ' ' + k)  
        item_total += v  
    print("Total number of items: " + str(item_total))  
  
displayInventory(stuff)
```

### Eine Funktion zum Hinzufügen von Listeninhalten zum Inventar-Dictionary

Nehmen Sie an, der Hort eines besieгten Drachens in dem Spiel wird wie folgt als Liste von Strings dargestellt:

```
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
```

Schreiben Sie eine Funktion namens `addToInventory(inventory, addedItems)`, wobei der Parameter `inventory` das Dictionary mit dem Inventar eines Spielers (wie im vorherigen Projekt) ist und der Parameter `addedItems` eine Liste wie `dragonLoot`.

Die Funktion `addToInventory()` soll ein Dictionary zurückgeben, das für das aktualisierte Inventar steht. Beachten Sie, dass die Liste `addedItems` mehrere Vorkommen des gleichen Gegenstands enthalten kann. Ihr Code sollte folgenden allgemeinen Aufbau haben:

```
def addToInventory(inventory, addedItems):
    # Bauen Sie hier Ihren eigenen Code ein

    inv = {'gold coin': 42, 'rope': 1}
    dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
    inv = addToInventory(inv, dragonLoot)
    displayInventory(inv)
```

Das vorstehende Programm sollte (mit der Funktion `displayInventory()` aus dem vorherigen Projekt) folgende Ausgabe hervorrufen:

```
Inventory:
45 gold coin
1 rope
1 ruby
1 dagger

Total number of items: 48
```



# 6

## Stringbearbeitung



Text gehört zu den häufigsten Formen von Daten, mit denen Ihre Programme umgehen müssen. Sie wissen bereits, wie Sie zwei Strings mit dem Operator + verketten, aber Sie können noch viel mehr tun als das. Es ist möglich, aus einem String Teilstrings zu entnehmen, Abstände hinzuzufügen oder zu entfernen, Buchstaben in Klein- oder Großbuchstaben umzuwandeln und zu überprüfen, ob die Strings korrekt formatiert sind. Es ist sogar möglich, Python-Code zu schreiben, um auf die Zwischenablage zum Kopieren und Einfügen von Text zuzugreifen.

In diesem Kapitel lernen Sie all das und noch viel mehr. Im Anschluss daran arbeiten Sie zwei Programmierprojekte durch, einen einfachen Passwortmanager und ein Programm, mit dem Sie die ermüdende Aufgabe automatisieren, Text zu formatieren.

### Umgang mit Strings

Als Erstes sehen wir uns an, wie Sie mit Python Strings schreiben, ausgeben und lesen.

## Stringliterale

Stringwerte in Python-Code einzugeben, ist ziemlich einfach: Sie beginnen und enden mit einem einfachen Anführungszeichen. Was aber, wenn innerhalb des Strings ein einfaches Anführungszeichen (bzw. Apostroph) steht. Eine Eingabe wie 'That is Alice's cat.' funktioniert nicht, da Python der Meinung ist, dass der String hinter Alice endet und der Rest (' cat.') kein gültiger Python-Code ist. Zum Glück gibt es mehrere Möglichkeiten, um Strings zu schreiben.

### Doppelte Anführungszeichen

Strings können genauso gut mit doppelten wie mit einfachen Anführungszeichen beginnen und enden. Ein Vorteil der Verwendung von doppelten Anführungszeichen besteht darin, dass im String dann einfache Anführungszeichen (bzw. Apostrophe) auftreten dürfen wie im folgenden Beispiel:

```
>>> spam = "That is Alice's cat."
```

Da der String mit einem doppelten Anführungszeichen beginnt, weiß Python, dass das einfache Anführungszeichen zu dem String gehört und nicht etwa dessen Ende kennzeichnet. Wenn Sie innerhalb des Strings jedoch sowohl einfache als auch doppelte Anführungszeichen verwenden, müssen Sie auf Maskierungszeichen zurückgreifen.

### Maskierungszeichen

Mithilfe eines *Maskierungszeichens* können Sie Zeichen in Strings aufnehmen, die sonst nicht zulässig wären. Als Maskierungszeichen wird der umgekehrte Schrägstrich oder *Backslash* verwendet (\), auf den dann das zu maskierende Zeichen folgt. Die komplette Maskierungssequenz für ein einfaches Anführungszeichen lautet also \'. Diese Zeichenfolge können Sie innerhalb eines Strings verwenden, der mit einfachen Anführungszeichen beginnt und endet. Um zu sehen, wie das funktioniert, geben Sie folgendes Beispiel in die interaktive Shell ein:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Da dem einfachen Anführungszeichen in Bob\'s ein Backslash vorausgeht, weiß Python, dass es nicht dazu dient, den String zu beenden. Mit den Maskierungssequenzen \' und \" können Sie daher einzelne bzw. doppelte Anführungszeichen in Strings einbauen.

Tabelle 6–1 führt die gängigen Maskierungssequenzen auf.

Maskierungssequenz	Ausgabe
\'	Einfaches Anführungszeichen
\"	Doppeltes Anführungszeichen
\t	Tabulator
\n	Zeilenumbruch
\\"	Backslash

**Tab. 6–1** Maskierungssequenzen

Geben Sie zum Ausprobieren Folgendes in die interaktive Shell ein:

```
>>> print("Hello there!\nHow are you?\nI'm doing fine.")
Hello there!
How are you?
I'm doing fine.
```

## Rohstrings

Wenn Sie vor das öffnende Anführungszeichen ein r stellen, kennzeichnen Sie den String damit als *Rohstring*. Dadurch werden alle Backslashes so ausgegeben, wie sie in dem String stehen. Geben Sie zur Veranschaulichung Folgendes in die interaktive Shell ein:

```
>>> print(r'That is Carol\'s cat.')
That is Carol's cat.
```

Da es sich um einen Rohstring handelt, nutzt Python den Backslash zwar weiterhin als Maskierungszeichen, sieht ihn gleichzeitig aber auch als Teil des Strings an. Rohstrings sind sehr nützlich, wenn Sie Stringwerte mit vielen Backslashes eingeben, z. B. Strings für reguläre Ausdrücke, wie sie im nächsten Kapitel beschrieben werden.

## Mehrzeilige Strings mit dreifachen Anführungszeichen

Mit der Maskierungssequenz \n können Sie einen Zeilenumbruch in einen String einschalten, aber meistens ist es einfacher, mehrzeilige Strings zu verwenden. Solche Strings in Python beginnen und enden mit drei einfachen oder drei doppelten Anführungszeichen. Alle Anführungszeichen, Tabulatoren und Zeilenumbrüche zwischen diesen dreifachen Anführungszeichen gelten als Teil des Strings. Die Einrückungsregeln für Blöcke gelten nicht für die Zeilen eines mehrzeiligen Strings.

Geben Sie im Dateieditor Folgendes ein:

```
print('''Dear Alice,  
Eve's cat has been arrested for catnapping, cat burglary, and extortion.  
Sincerely,  
Bob''')
```

Speichern Sie dieses Programm als *catnapping.py* und führen Sie es aus. Sie erhalten folgende Ausgabe:

```
Dear Alice,  
Eve's cat has been arrested for catnapping, cat burglary, and extortion.  
Sincerely,  
Bob
```

Beachten Sie, dass das einzelne Anführungszeichen in Eve's nicht maskiert werden muss. In mehrzeiligen Strings ist die Maskierung von einzelnen und doppelten Anführungszeichen optional. Der folgende Aufruf von `print()` gibt den gleichen Text aus, verwendet aber nicht die Schreibweise als mehrzeiliger String:

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping, cat  
burglary, and extortion.\n\nSincerely,\nBob')
```

## Mehrzeilige Kommentare

Das Doppelkreuz (#) kennzeichnet den Beginn eines Kommentars, der den ganzen Rest der Zeile einnimmt. Für Kommentare, die mehrere Zeilen überspannen, werden jedoch häufig mehrzeilige Strings verwendet. Der folgende Beispiel ist gültiger Python-Code:

```
"""This is a test Python program.  
Written by Al Sweigart al@inventwithpython.com  
  
This program was designed for Python 3, not Python 2.  
"""  
  
def spam():  
    """This is a multiline comment to help  
    explain what the spam() function does."""  
    print('Hello!')
```

## Strings indizieren und Slices entnehmen

Für Strings können Sie Indizes und Slices genauso verwenden wie für Listen. Eine String wie 'Hello world!' lässt sich auch als eine Liste vorstellen, wobei jedes Zeichen ein Element mit einem zugehörigen Index ist:

```
' H   e   l   l   o       w   o   r   l   d   !   '
  0   1   2   3   4   5   6   7   8   9   10  11
```

Das Leerzeichen und das Ausrufezeichen werden bei der Anzahl der Zeichen berücksichtigt. 'Hello world!' ist also zwölf Zeichen lang, von H an Index 0 bis zum ! an Index 11.

Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

Wenn Sie einen Index angeben, erhalten Sie das Zeichen an der entsprechenden Position im String. Geben Sie dabei einen Bereich von einer Indexposition zu einer anderen an, so ist der Startindex in diesem Bereich eingeschlossen, der Endindex aber nicht. Aus diesem Grund gibt `spam[0:5]` in dem Beispiel 'Hello' zurück: Der Teilstring `spam[0:5]` schließt alles von `spam[0]` bis einschließlich `spam[spam4]` ein, aber nicht mehr das Leerzeichen am Index 5.

Wenn Sie einen Slice aus einem String erstellen, wird der ursprüngliche String dadurch nicht verändert. Sie können einem String in einer Variablen einen Slice entnehmen und diesen in einer eigenen Variablen speichern. Das können Sie in der interaktiven Shell wie folgt ausprobieren:

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

Durch den Slice-Vorgang und die Speicherung des resultierenden Teilstrings in einer anderen Variablen haben Sie sowohl den kompletten String als auch den Teilstring griffbereit.

### Die Operatoren `in` und `not in` für Strings

Die Operatoren `in` und `not in` können Sie für Strings ebenso verwenden wie für Listenwerte. Ein Ausdruck, in dem Sie zwei Strings mit `in` oder `not in` verknüpfen, wird zu dem booleschen Wert `True` oder `False` ausgewertet. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> 'Hello' in 'Hello World'  
True  
>>> 'Hello' in 'Hello'  
True  
>>> 'HELLO' in 'Hello World'  
False  
>>> '' in 'spam'  
True  
>>> 'cats' not in 'cats and dogs'  
False
```

Mit diesen Ausdrücken prüfen Sie, ob der erste String (und zwar der genaue String unter Berücksichtigung der Groß- und Kleinschreibung) in dem zweiten String enthalten ist.

### Nützliche Stringmethoden

Es gibt verschiedene Stringmethoden, um Strings zu analysieren oder um Stringwerte zu transformieren. In diesem Abschnitt beschreibe ich die Methoden, die Sie am häufigsten benötigen werden.

#### Die Stringmethoden `upper()`, `lower()`, `isupper()` und `islower()`

Die Stringmethoden `upper()` und `lower()` geben einen neuen String zurück, bei dem alle Buchstaben des Originalstrings in Groß- bzw. Kleinbuchstaben verwandelt wurden. Zeichen, die keine Buchstaben sind, bleiben unverändert. Probieren Sie das folgende Beispiel in der interaktiven Shell aus:

```
>>> spam = 'Hello world!'  
>>> spam = spam.upper()  
>>> spam  
'HELLO WORLD!'
```

```
>>> spam = spam.lower()  
>>> spam  
'hello world!'
```

Beachten Sie, dass diese Methoden den String selbst nicht ändern, sondern einen neuen Stringwert zurückgeben. Um den Originalstring zu bearbeiten, müssen Sie den neuen String der Variablen zuweisen, in der der ursprüngliche String gespeichert ist. Aus diesem Grunde steht in dem obigen Beispiel `spam = spam.upper()` statt einfach `spam.upper()`. (Das ist genauso wie bei arithmetischen Operationen mit Variablen. Wenn die Variable `eggs` einen numerischen Wert enthält, dann ändert `eggs + 3` diesen Wert nicht; dazu müssen Sie `eggs = eggs + 3` schreiben.)

Die Methoden `upper()` und `lower()` sind vor allem dann sehr nützlich, wenn Sie Strings ohne Berücksichtigung der Groß- und Kleinschreibung vergleichen wollen. So sind beispielsweise die Strings '`great`' und '`GREAT`' nicht gleich. In dem folgenden kleinen Programm kommt es jedoch nicht darauf an, ob der Benutzer `Great`, `GREAT` oder `grEAT` eingibt, da der String zunächst komplett in Kleinbuchstaben umgewandelt wird.

```
print('How are you?')  
feeling = input()  
if feeling.lower() == 'great':  
    print('I feel great too.')  
else:  
    print('I hope the rest of your day is good.')
```

Wenn Sie dieses Programm ausführen, wird zunächst die Frage angezeigt. Wenn der Benutzer irgendeine Variante von `great` eingibt, z. B. `GREAT`, dann lautet die Ausgabe `I feel great too.` Durch solchen zusätzlichen Code, der auch mit unterschiedlichen oder gar falschen Schreibweisen von Benutzereingaben zurechtkommt, lassen sich Ihre Programme einfacher nutzen und Sie sind weniger fehleranfällig.

```
How are you?  
GREAT  
I feel great too.
```

Die Methoden `isupper()` und `islower()` geben den booleschen Wert `True` zurück, wenn der String mindestens einen Buchstaben enthält und sämtliche Buchstaben Groß- bzw. Kleinbuchstaben sind, anderenfalls `False`. Beachten Sie die Ausgabe der einzelnen Methodenaufrufe in den folgenden Beispielen:

```
>>> spam = 'Hello world!'  
>>> spam.islower()  
False
```

```
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

Da `upper()` und `lower()`Strings zurückgeben, können Sie auf ihre Rückgabewerte wiederum Stringmethoden anwenden. Die entsprechenden Ausdrücke sehen aus wie eine Kette von Methodenaufrufen:

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

## Die `isX`-Stringmethoden

Neben `islower()` und `isupper()` gibt es noch weitere Stringmethoden, deren Namen mit `is` beginnen. Sie alle geben einen booleschen Wert zurück, der die Natur des Strings beschreibt. Besonders gebräuchlich sind die folgenden `isX`-Stringmethoden:

- `isalpha()` gibt `True` zurück, wenn der String nur aus Buchstaben besteht.
- `isalnum()` gibt `True` zurück, wenn der String nur aus Buchstaben und Zahlen besteht.
- `isdecimal()` gibt `True` zurück, wenn der String nur aus numerischen Zeichen besteht.
- `isspace()` gibt `True` zurück, wenn der String nur aus Leerzeichen, Tabulatoren und Zeilenumbrüchen besteht.
- `istitle()` gibt `True` zurück, wenn alle Wörter in dem String mit einem Großbuchstaben beginnen und ansonsten nur Kleinbuchstaben enthalten.

Probieren Sie in der interaktiven Shell Folgendes aus:

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> ' '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

Die `isX`-Methoden sind praktisch zur Validierung von Benutzereingaben. Das folgende Beispielprogramm fragt die Benutzer wiederholt nach ihrem Alter und einem Passwort, bis sie eine gültige Antwort eingegeben haben. Geben Sie dieses Programm im Dateieditor ein und speichern Sie es als `validateInput.py`:

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters and numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters and numbers.')
```

In der ersten `while`-Schleife fragen wir den Benutzer nach dem Alter und speichern die Eingabe in `age`. Wenn `age` ein gültiger (dezimaler) Wert ist, verlassen wir mit `break` die erste `while`-Schleife und gehen zur zweiten über, in der wir nach einem Passwort fragen. Andernfalls weisen wir den Benutzer darauf hin, dass er eine

Zahl eingeben muss, und fordern ihn erneut auf, sein Alter zu nennen. In der zweiten while-Schleife fragen wir nach einem Passwort. Ist die Eingabe alphanumerisch, speichern wir sie in password und verlassen die Schleife. Wenn nicht, weisen wir den Benutzer darauf hin, dass das Passwort alphanumerisch sein muss, und bitten ihn erneut um Eingabe.

Die Ausgabe dieses Programms sieht wie folgt aus:

```
Enter your age:  
forty two  
Please enter a number for your age.  
Enter your age:  
42  
Select a new password (letters and numbers only):  
secr3t!  
Passwords can only have letters and numbers.  
Select a new password (letters and numbers only):  
secr3t
```

Mit dem Aufruf von `isdecimal()` bzw. `isalnum()` für die Variablen können wir prüfen, ob die gespeicherten Werte dezimal bzw. alphanumerisch sind oder nicht. Damit können wir hier Eingaben wie `forty two` bzw. `secr3t!` verwerfen und `42` bzw. `secr3t` akzeptieren.

### Die Stringmethoden `startswith()` und `endswith()`

Die Methoden `startswith()` und `endswith()` geben `True` zurück, wenn der Stringwert, für den sie aufgerufen wurden, mit dem übergebenen String beginnt bzw. endet. Andernfalls geben sie `False` zurück. Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> 'Hello world!'.startswith('Hello')  
True  
>>> 'Hello world!'.endswith('world!')  
True  
>>> 'abc123'.startswith('abcdef')  
False  
>>> 'abc123'.endswith('12')  
False  
>>> 'Hello world!'.startswith('Hello world!')  
True  
>>> 'Hello world!'.endswith('Hello world!')  
True
```

Diese Methoden sind praktische Alternativen zum Gleichheitsoperator `==`, wenn Sie nicht den gesamten String, sondern nur den ersten oder letzten Teil auf Gleichheit mit einem anderen String überprüfen müssen.

## Die Methoden `join()` und `split()`

Wenn Sie mehrere Strings zu einem einzelnen Stringwert verbinden möchten, rufen Sie die Methode `join()` für den String auf, der als Trennzeichen dienen soll, und übergeben ihr die Liste der zu verknüpfenden Strings. Der Rückgabewert ist wiederum ein String, und zwar die Verkettung der Strings in der Liste:

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

Der String, für den `join()` aufgerufen wird, erscheint dabei zwischen den einzelnen Strings der als Argument übergebenen Liste. Wenn Sie `join(['cats', 'rats', 'bats'])` für den String `', '` aufrufen, wird daher `'cats, rats, bats'` zurückgegeben.

Merken Sie sich, dass Sie `join()` für das Trennzeichen aufrufen und ihr den Listenwert übergeben müssen und nicht umgekehrt (was ein häufiger Fehler ist). Die Methode `split()` bewirkt das Gegenteil. Sie wird für einen Stringwert aufgerufen und gibt eine Liste von Teilstrings zurück:

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

Standardmäßig wird der String (hier `'My name is Simon'`) an den Stellen aufgeteilt, an denen sich ein Weißraumzeichen wie ein Leerzeichen, ein Tabulator oder ein Zeilenumbruch befindet. Diese Weißraumzeichen sind in den Strings der zurückgegebenen Liste jedoch nicht enthalten. Um eine andere Aufteilung zu bewirken, können Sie der Methode `split()` auch ein Trennzeichen übergeben. Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']
>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

Häufig wird `split()` dazu verwendet, einen mehrzeiligen String an den Zeilenumbrüchen zu trennen. Das können Sie in der interaktiven Shell wie folgt ausprobieren:

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment".
```

```

Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a container in the
fridge', 'that is labeled "Milk Experiment".', '', 'Please do not drink it.',
'Sincerely,', 'Bob']

```

Hier übergeben Sie `split()` das Argument `'\n'`. Dadurch trennt die Methode den in `spam` gespeicherten mehrzeiligen String an den Zeilenumbrüchen auf und gibt eine Liste zurück, deren einzelne Elemente jeweils eine Zeile des ursprünglichen Strings darstellen.

### Text mit `rjust()`, `ljust()` und `center()` ausrichten

Die Methoden `rjust()` und `ljust()` werden für einen String aufgerufen und geben eine Variante dieses Strings zurück, bei der der Text mithilfe von Leerzeichen ausgerichtet ist. Das erste Argument beider Methoden ist ein Integer, der die Gesamtlänge des ausgerichteten Strings angibt. Das können Sie in der interaktiven Shell wie folgt ausprobieren:

```

>>> 'Hello'.rjust(10)
'      Hello'
>>> 'Hello'.rjust(20)
'                      Hello'
>>> 'Hello World'.rjust(20)
'          Hello World'
>>> 'Hello'.ljust(10)
'Hello      '

```

`'Hello'.rjust(10)` bedeutet, dass `'Hello'` in einem String der Gesamtlänge 10 rechtsbündig ausgerichtet werden soll. Da `'Hello'` selbst fünf Zeichen umfasst, müssen links davon fünf Leerzeichen eingeschaltet werden, um die angegebene Gesamtlänge zu erreichen.

In einem optionalen zweiten Argument können Sie in `rjust()` und `ljust()` ein anderes Füllzeichen als das Leerzeichen angeben:

```

>>> 'Hello'.rjust(20, '*')
*****Hello
>>> 'Hello'.ljust(20, '-')
Hello-----

```

Die Stringmethode `center()` dient ebenso wie `ljust()` und `rjust()` zur Ausrichtung, zentriert den Text aber:

```
>>> 'Hello'.center(20)
      Hello
>>> 'Hello'.center(20, '=')
'=====Hello====='
```

Diese Methoden sind dann nützlich, wenn Sie Daten tabellarisch und mit den richtigen Abständen ausgeben möchten. Geben Sie den folgenden Code in den Dateieditor ein und speichern Sie ihn als *picnicTable.py*:

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))

picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

In diesem Programm definieren wir die Methode `printPicnic()`, die ein Dictionary entgegennimmt und die darin enthaltenen Informationen mithilfe von `center()`, `ljust()` und `rjust()` in einem übersichtlichen Tabellenformat ausgibt.

Das Dictionary, das wir `printPicnic()` übergeben, heißt `picnicItems` und enthält vier Sandwiches, zwölf Äpfel, vier Tassen und 8000 Kekse. Diese Informationen wollen wir in zwei Spalten ausgeben, wobei die Bezeichnung auf der linken und die Menge auf der rechten Seite steht.

Dazu müssen wir festlegen, wie breit die beiden Spalten jeweils sein sollen. Diese Werte übergeben wir zusammen mit dem Dictionary an `printPicnic()`.

Die Methode `printPicnic()` nimmt ein Dictionary, die Breite `leftWidth` der linken Spalte und die Breite `rightWidth` der rechten Spalte entgegen. Als Erstes gibt Sie den Tabellentitel `PICNIC ITEMS` zentriert aus. Anschließend durchläuft sie das Dictionary und gibt jedes Schlüssel-Wert-Paar in einer Tabellenzeile aus, wobei der Schlüssel linksbündig und mit Punkten aufgefüllt auf der linken Seite steht und der Wert rechtsbündig und mit Leerzeichen aufgefüllt auf der rechten Seite.

Nach der Definition von `printPicnic()` geben wir `picnicItems` an und rufen die Methode zweimal auf, wobei wir ihr jeweils unterschiedliche Werte für die Spaltenbreiten übergeben.

Wenn Sie dieses Programm ausführen, werden die Picknickutensilien zweimal ausgegeben. In der ersten Fassung ist die linke Spalte zwölf und die rechte fünf Zeichen breit, in der zweiten Fassung sind es sechzig bzw. sechs Zeichen:

```
--PICNIC ITEMS--
sandwiches..     4
apples.....    12
cups.....      4
cookies..... 8000
```

```
-----PICNIC ITEMS-----
sandwiches..... 4
apples..... 12
cups..... 4
cookies..... 8000
```

Mit `rjust()`, `ljust()` und `center()` können Sie Strings übersichtlich ausrichten, auch wenn Sie nicht wissen, wie lang die Strings im Einzelnen sind.

### Weißraum mit `strip()`, `rstrip()` und `lstrip()` entfernen

Es kann vorkommen, dass Sie Weißraumzeichen (Leerzeichen, Tabulatoren oder Zeilenumbrüche) von der linken, der rechten oder beiden Seiten eines Strings entfernen müssen. Die Methode `strip()` gibt einen neuen String zurück, der weder am Anfang noch am Ende Weißraumzeichen aufweist, während `lstrip()` und `rstrip()` nur Weißraumzeichen vom linken bzw. rechten Ende entfernen:

```
>>> spam = 'Hello World '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World '
>>> spam.rstrip()
'Hello World'
```

Optional können Sie mit einem Stringargument angeben, welche Zeichen an den Enden entfernt werden sollen:

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Das Argument 'ampS' weist `strip()` an, jegliche Vorkommen von a, m, p und S vom Ende des in `spam` gespeicherten Strings zu entfernen. Die Reihenfolge der Zeichen in dem an `strip()` übergebenen Argumentstring spielt dabei keine Rolle: `strip('ampS')` hat die gleiche Auswirkung wie `strip('mapS')` oder `strip('Spam')`.

### Strings mit dem Modul `pyperclip` kopieren und einfügen

Das Modul `pyperclip` enthält die Funktionen `copy()` und `paste()`, die Text an die Zwischenablage des Computers senden bzw. von dort empfangen können. Wenn Sie die Ausgabe Ihres Programms in die Zwischenablage stellen, können Sie sie von dort aus einfacher in eine E-Mail, ein Textverarbeitungssystem oder eine andere Software einfügen.

Pyperclip ist im Lieferumfang von Python nicht enthalten. Um es zu installieren, folgen Sie der Anleitung zur Installation von Drittanbietermodulen aus Anhang A. Anschließend geben Sie in der interaktiven Shell Folgendes ein:

```
>>> import pyperclip  
>>> pyperclip.copy('Hello world!')  
>>> pyperclip.paste()  
'Hello world!'
```

Wenn irgendwelcher anderer Code außerhalb Ihres Programms den Inhalt der Zwischenablage ändert, gibt die Funktion `paste()` natürlich diesen neuen Inhalt zurück. Wenn ich beispielsweise in einem Textverarbeitungsprogramm den Satz *For example, if I copied this sentence to the clipboard and then called paste(), it would look like this:* in die Zwischenablage eingebe und dann `paste()` aufrufe, erhalte ich folgendes Ergebnis:

```
>>> pyperclip.paste()  
'For example, if I copied this sentence to the clipboard and then called  
paste(), it would look like this:'
```

### Python-Skripte außerhalb von IDLE ausführen

Bis jetzt haben Sie Python-Skripte nur in der interaktiven Shell und dem Dateieditor von IDLE ausgeführt. Allerdings wollen Sie sicherlich nicht jedes Mal erst IDLE starten und darin das gewünschte Skript öffnen, wenn Sie ein Python-Programm ausführen möchten. Zum Glück gibt es auch einfachere Möglichkeiten. Die erforderlichen Schritte unterscheiden sich je nachdem, ob Sie Windows, OS X oder Linux verwenden. Eine genaue Beschreibung der Vorgehensweise erhalten Sie in Anhang B. Dort erfahren Sie nicht nur, wie Sie Python-Skripte auf komfortable Weise ausführen, sondern auch, wie Sie Befehlszeilenargumente übergeben können (was nicht möglich ist, wenn Sie die Programme in IDLE ausführen).

## Projekt: Passwortsafe

Wenn Sie Konten auf verschiedenen Websites haben, ist es gefährlich, für alle dasselbe Passwort zu verwenden, denn wenn Hacker eine dieser Websites knacken, haben sie damit auch die Passwörter für all Ihre anderen Konten. Am besten ist es, einen Passwortmanager auf Ihrem Computer zu verwenden. Sie müssen sich dann nur ein einziges Masterpasswort merken, um auf den Passwortmanager zuzugreifen. Anschließend können Sie die darin gespeicherten Passwörter für

beliebige Konten in die Zwischenablage übertragen und von dort aus in das Passwortfeld der jeweiligen Website einfügen.

### Die Projekte in den Kapiteln

Dies ist das erste Projekt in diesem Buch. Von nun an finden Sie in jedem Kapitel Projekte zur Veranschaulichung des besprochenen Stoffs. Dabei beginnen Sie jeweils in einem leeren Dateieditorfenster und haben am Ende ein komplettes, funktionierendes Programm. Wie bei den Beispielen für die interaktive Shell sollten Sie die Projektabschnitte nicht nur lesen, sondern auch auf Ihrem Computer nachvollziehen.

## Schritt 1: Programmdesign und Datenstrukturen

Bei der Ausführung dieses Programms soll die Bezeichnung für das gewünschte Konto – z. B. *email* oder *blog* – als Befehlszeilenargument übergeben werden. Das Passwort für dieses Konto wird dann in die Zwischenablage kopiert, sodass der Benutzer es von dort aus in das Passwortfeld einfügen kann. Dadurch kann der Benutzer lange, komplizierte Passwörter verwenden, ohne sie sich merken zu müssen.

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das Programm als *pw.py*. Die erste Zeile des Programms muss mit der Zeichenfolge `#!` (»Shebang«) beginnen (siehe Anhang B). Außerdem sollten Sie einen Kommentar angeben, der eine kurze Beschreibung des Programms enthält. Da Sie die einzelnen Kontonamen mit dem jeweiligen Passwort verknüpfen möchten, können Sie diese Strings in einem Dictionary speichern. Das Programm sieht also zunächst wie folgt aus:

```
#! python3
# pw.py - Ein unsicherer Passwortsafe.

PASSWORDS = {'email': 'F7min1BDDuvMJuxESSKHFhTxFtjVB6',
              'blog': 'VmALvQyKAxViH5G8v01if1MLZF3sdt',
              'luggage': '12345'}
```

## Schritt 2: Befehlszeilenargumente verarbeiten

Die Befehlszeilenargumente werden in der Variablen `sys.argv` gespeichert. (Weitere Informationen über die Verwendung von Befehlszeilenargumenten in Ihren Programmen erhalten Sie in Anhang B.) Das erste Element in der Liste `sys.argv` muss immer der String für den Dateinamen des Programms sein ('`pw.py`'), das

zweite das erste Befehlszeilenargument. Bei diesem Programm ist das Argument die Bezeichnung des Kontos, dessen Passwort Sie abrufen wollen. Da das Befehlszeilenargument obligatorisch ist, geben Sie eine erklärende Meldung an den Benutzer aus, wenn er die Angabe dieses Arguments vergessen hat (d. h., wenn die Liste sys.argv weniger als zwei Elemente enthält). Ergänzen Sie das Programm wie folgt:

```
#! python3
# pw.py - Ein unsicherer Passwortsafe.

PASSWORDS = {'email': 'F7min1BDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}

import sys
if len(sys.argv) < 2:
    print('Usage: python pw.py [account] - copy account password')
    sys.exit()

account = sys.argv[1]      # Das erste Befehlszeilenargument ist der Kontoname
```

### Schritt 3: Das richtige Passwort kopieren

Der Kontoname ist nun als String in der Variablen account gespeichert. Jetzt müssen Sie herausfinden, ob diese Bezeichnung auch als Schlüssel im Dictionary PASSWORDS vorkommt. Wenn ja, kopieren Sie den Wert dieses Schlüssels mit pyperclip.copy() in die Zwischenablage. (Da Sie das Modul pyperclip verwenden, müssen Sie es importieren.) Beachten Sie, dass die Variable account nicht unbedingt erforderlich ist, denn statt account könnten Sie überall im Programm auch sys.argv[1] verwenden. Ein Variablenname wie account macht den Code aber viel verständlicher als eine kryptische Angabe wie sys.argv[1].

Ergänzen Sie das Programm wie folgt:

```
#! python3
# pw.py - Ein unsicherer Passwortsafe.

PASSWORDS = {'email': 'F7min1BDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}

import sys, pyperclip
if len(sys.argv) < 2:
    print('Usage: python pw.py [account] - copy account password')
    sys.exit()
```

```

account = sys.argv[1]      # Das erste Befehlszeilenargument ist der Kontoname

if account in PASSWORDS:
    pyperclip.copy(PASSWORDS[account])
    print('Password for ' + account + ' copied to clipboard.')
else:
    print('There is no account named ' + account)

```

Der neu hinzugekommene Code sucht im Dictionary `PASSWORDS` nach dem Kontonamen. Ist dieser Name als Schlüssel in dem Dictionary vorhanden, rufen wir den zugehörigen Wert ab, kopieren ihn in die Zwischenablage und geben eine entsprechende Meldung aus. Andernfalls erscheint die Meldung, dass es kein Konto dieses Namens gibt.

Das ist auch schon das vollständige Skript. Mit den Anweisungen aus Anhang B zum Starten von Befehlszeilenprogrammen haben Sie nun eine schnelle Möglichkeit, um die Passwörter Ihrer Konten in die Zwischenablage zu kopieren. Wenn Sie das Programm um ein neues Passwort ergänzen wollen, müssen Sie das Dictionary `PASSWORDS` im Quellcode ergänzen.

Natürlich sollten Sie Ihre gesammelten Passwörter nicht alle an einer zentralen Stelle speichern, die jeder ganz einfach kopieren kann. Allerdings können Sie dieses Programm auch abwandeln und etwa dazu benutzen, normalen Text rasch in die Zwischenablage zu kopieren. Nehmen wir beispielsweise an, Sie senden E-Mails, in denen oft die gleichen Standardabsätze vorkommen. In diesem Fall können Sie diese Absätze als Werte in das Dictionary aufnehmen (dem Sie für diesen Zweck am besten einen anderen Namen geben) und dann schnell die gewünschten Texte auswählen und in die Zwischenablage übertragen.

Unter Windows können Sie eine Batchdatei erstellen, um das Programm vom *Ausführen*-Fenster (`Windows` + `R`) aus starten zu können. (Mehr über Batchdateien erfahren Sie in Anhang B.) Geben Sie im Dateieditor Folgendes ein und speichern Sie die Datei als `pw.bat` im Ordner `C:\Windows`:

```

@py.exe C:\Python34\pw.py %*
@pause

```

Nun können Sie das Passwortsafeprogramm in Windows einfach dadurch starten, dass Sie `Windows` + `R` drücken und `pw <kontoname>` eingeben.

## **Projekt: Aufzählungspunkte zu einem Wiki-Markup hinzufügen**

Bei der Bearbeitung eines Wikipedia-Artikels können Sie eine Spiegelstrichaufzählung, also eine Liste mit Aufzählungspunkten erstellen, indem Sie jeden Listeneintrag in eine eigene Zeile schreiben und ihm ein Sternchen voranstellen. Wenn Sie eine

sehr lange Liste haben, könnten Sie all diese Sternchen natürlich ebenfalls manuell an den Anfang jeder Zeile setzen, eines nach dem anderen. Mit dem folgenden kurzen Python-Skript können Sie die Aufgabe aber auch automatisieren.

Das Skript *bulletPointAdder.py* ruft den Text von der Zwischenablage ab, fügt am Anfang jeder Zeile ein Sternchen und ein Leerzeichen hinzu und kopiert den neuen Text in die Zwischenablage. Nehmen wir beispielsweise an, Sie haben den folgenden Text (für den Wikipedia-Artikel »Listen von Listen von Listen«) in die Zwischenablage übertragen:

```
Lists of animals
Lists of aquarium life
Lists of biologists by author abbreviation
Lists of cultivars
```

Wenn Sie nun *bulletPointAdder.py* ausführen, enthält die Zwischenablage anschließend folgenden Text:

```
* Lists of animals
* Lists of aquarium life
* Lists of biologists by author abbreviation
* Lists of cultivars
```

Diesen Text mit vorangestellten Sternchen können Sie nun ganz einfach als Spiegelstrichaufzählung in einen Wikipedia-Artikel kopieren.

### Schritt 1: Text von und zur Zwischenablage übertragen

Das Programm *bulletPointAdder.py* soll Folgendes tun können:

1. Text aus der Zwischenablage einfügen
2. Den Text auf irgendeine Weise bearbeiten
3. Den neuen Text in die Zwischenablage kopieren

Der zweite Schritt ist ein bisschen knifflig, für Schritt 1 und 3 dagegen müssen Sie lediglich `pyperclip.copy()` bzw. `pyperclip.paste()` aufrufen. Als Erstes schreiben wir den Teil des Programms, der diese beiden Schritte abdeckt. Geben Sie folgenden Code ein und speichern Sie das Programm als *bulletPointAdder.py*:

```
#! python3
# bulletPointAdder.py - Fügt Wikipedia-Aufzählungszeichen zu Beginn jeder
# Zeile des Textes in der Zwischenablage hinzu.

import pyperclip
text = pyperclip.paste()
```

```
# TODO: Zeilen trennen und Sternchen hinzufügen.  
pyperclip.copy(text)
```

Der TODO-Kommentar dient als Erinnerungsstütze dafür, dass Sie diesen Teil des Programms noch ergänzen müssen. Genau das wollen wir im nächsten Schritt erledigen.

## Schritt 2: Textzeilen trennen und Sternchen hinzufügen

Der Aufruf von `pyperclip.paste()` gibt den gesamten Text in der Zwischenablage als einen einzigen, langen String zurück. In dem zuvor genannten Beispiel für die »Liste von Listen von Listen« sieht der String in `text` wie folgt aus:

```
'Lists of animals\nLists of aquarium life\nLists of biologists by author  
abbreviation\nLists of cultivars'
```

Aufgrund des Zeilenumbruchzeichens `\n` wird der String bei der Bildschirmausgabe oder beim Einfügen aus der Zwischenablage in mehreren Zeilen wiedergegeben. Dieser Stringwert enthält in Wirklichkeit mehrere Zeilen, vor die wir jeweils ein Sternchen setzen wollen.

Sie könnten Code schreiben, der alle Zeilenumbruchzeichen (`\n`) in diesem String sucht und dahinter jeweils ein Sternchen platziert. Einfacher ist es jedoch, mithilfe der Methode `split()` eine Liste von Strings zurückzugeben, die jeweils für eine Zeile des Originalstrings stehen, und den einzelnen Einträgen in dieser Liste jeweils ein Sternchen voranzustellen.

Ergänzen Sie das Programm wie folgt:

```
#! python3  
# bulletPointAdder.py - Fügt Wikipedia-Aufzählungszeichen zu Beginn jeder  
# Zeile des Textes in der Zwischenablage hinzu.  
  
import pyperclip  
text = pyperclip.paste()  
  
# Trennt Zeilen und fügt Sternchen hinzu  
lines = text.split('\n')  
for i in range(len(lines)):    # Durchläuft alle Indizes in der Liste "lines"  
    lines[i] = '*' + lines[i] # Fügt Stern zu allen Strings in "lines" hinzu  
  
pyperclip.copy(text)
```

Wir trennen den Text also an den Zeilenumbrüchen auf, um eine Liste zu bekommen, deren einzelne Einträge jeweils für eine Textzeile stehen. Diese Liste speichern

wir in `lines`. Anschließend durchlaufen wir die Elemente in `lines` in einer Schleife. Jeder Zeile stellen wir ein Sternchen und ein Leerzeichen voran. Nun beginnt jeder String in `lines` mit einem Sternchen.

### Schritt 3: Die veränderten Zeilen zusammenfügen

Die Liste `lines` enthält nun die veränderten Zeilen, die mit Sternchen beginnen. Allerdings erwartet `pyperclip.copy()` einen einzelnen Stringwert und keine Liste von Strings. Daher müssen wir `lines` an die Methode `join()` übergeben, um alle Einträge in der Liste wieder zu einem einzigen String zusammenzufügen. Ergänzen Sie das Programm wie folgt:

```
#! python3
# bulletPointAdder.py - Fügt Wikipedia-Aufzählungszeichen zu Beginn jeder
# Zeile des Textes in der Zwischenablage hinzu.

import pyperclip
text = pyperclip.paste()

# Trennt Zeilen und fügt Sternchen hinzu
lines = text.split('\n')
for i in range(len(lines)):
    # Durchläuft alle Indizes in der Liste "lines"
    lines[i] = '*' + lines[i] # Fügt Stern zu allen Strings in "lines" hinzu
text = '\n'.join(lines)
pyperclip.copy(text)
```

Das Programm ersetzt den Text in der Zwischenablage durch eine Version, die Sternchen zu Beginn jeder Zeile aufweist. Jetzt ist das Programm fertig und Sie können es mit jedem Text ausprobieren, den Sie in die Zwischenablage kopiert haben.

Es kann natürlich sein, dass Sie keinen Bedarf dafür haben, genau diese Aufgabe zu automatisieren. Aber es ist durchaus möglich, dass Sie irgendeine andere Form der Textbearbeitung durchführen möchten, beispielsweise abschließende Leerzeichen am Zeilenende entfernen oder Text von Groß- in Kleinbuchstaben umwandeln. Ganz gleich, was Sie tun müssen, Sie können die Zwischenablage zur Ein- und Ausgabe verwenden.

## Zusammenfassung

Text ist eine häufig auftretende Form von Daten, weshalb Python eine Menge hilfreicher Methoden zur Verarbeitung von Strings aufweist. Index- und Slice- sowie Stringmethoden werden Sie in praktisch jedem Ihrer Python-Programme verwenden.

Die Programme, die Sie zurzeit schreiben, mögen nicht sehr anspruchsvoll wirken – sie haben keine grafische Benutzeroberfläche mit Bildern und farbigem Text. Bis jetzt geben sie nur Text mit `print()` aus und nehmen Benutzereingaben mit `input()` entgegen. Allerdings können die Benutzer auch über die Zwischenablage viel Text eingeben. Damit haben wir eine praktische Möglichkeit zur Hand, um Programme zu schreiben, die große Textmengen bearbeiten. Textgestützte Programme haben zwar keine schicken Fenster und keine aufsehenerregende Grafik, können aber viel nützliche Arbeit in sehr kurzer Zeit erledigen.

Eine andere Möglichkeit, um große Mengen an Text zu bearbeiten, besteht darin, ihn direkt aus Dateien auf der Festplatte zu lesen oder dort zu schreiben. Wie Sie das mit Python erledigen, erfahren Sie im übernächsten Kapitel.

## Wiederholungsfragen

1. Was sind Maskierungssequenzen?
2. Wofür stehen die Maskierungssequenzen `\n` und `\t`?
3. Wie können Sie einen umgekehrten Schrägstrich (Backslash) in einen String einfügen?
4. "Howl's Moving Castle" ist ein gültiger Stringwert. Warum führt das als Apostroph verwendete einfache Anführungszeichen in `Howl's` nicht zu einem Problem, obwohl es nicht maskiert ist?
5. Wie können Sie einen String mit Zeilenumbrüchen schreiben, ohne darin die Zeichenfolge `\n` zu verwenden?
6. Wozu werden die folgenden Ausdrücke ausgewertet?
  - `'Hello world!'[1]`
  - `'Hello world!'[0:5]`
  - `'Hello world!'[:5]`
  - `'Hello world!'[3:]`
7. Wozu werden die folgenden Ausdrücke ausgewertet?
  - `'Hello'.upper()`
  - `'Hello'.upper().isupper()`
  - `'Hello'.upper().lower()`
8. Wozu werden die folgenden Ausdrücke ausgewertet?
  - `'Remember, remember, the fifth of November.'.split()`
  - `'-'.join('There can be only one.'.split())`

9. Mit welchen Stringmethoden können Sie einen String rechtsbündig, linksbündig oder zentriert ausrichten?
10. Wie können Sie Weißraumzeichen am Anfang oder Ende eines Strings abschneiden?

## Übungsprojekt

Schreiben Sie zur Übung ein Programm, das die folgende Aufgabe erledigt:

### Tabellenausgabe

Schreiben Sie eine Funktion namens `printTable()`, die eine Liste aus Stringlisten entgegennimmt und als übersichtlich gegliederte Tabelle mit rechtsbündigen Spalten ausgibt. Alle inneren Listen sollen die gleiche Anzahl von Strings aufweisen. Ein möglicher Wert sieht also beispielsweise wie folgt aus:

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],
             ['Alice', 'Bob', 'Carol', 'David'],
             ['dogs', 'cats', 'moose', 'goose']]
```

Die Funktion `printTable()` soll dies wie folgt ausgeben:

```
    apples Alice  dogs
    oranges   Bob  cats
    cherries Carol moose
      banana David goose
```

Tipp: Der Code muss zunächst den jeweils längsten String in allen inneren Listen finden, damit die Spalte breit genug gemacht werden kann, um alle Strings aufzunehmen. Die Höchstbreite der einzelnen Spalten können Sie als Integerliste speichern. Am Anfang der Funktion `printTable()` kann `colWidths[0] * len(tableData)` stehen. Dadurch erstellen Sie eine Liste, die genauso viele Einträge des Werts 0 enthält, wie es innere Listen in `tableData` gibt. Darin kann nun `colWidths[0]` die Breite des längsten Strings in `tableData[0]` aufnehmen, `colWidths[1]` die Breite des längsten Strings in `tableData[1]` usw. Um herauszufinden, welchen Integer Sie als Breite an die Methode `rjust()` übergeben müssen, ermitteln Sie den größten Wert in der Liste `colWidths`.



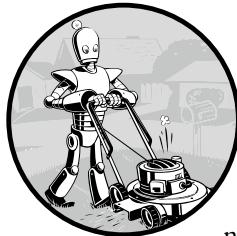
# Teil 2

**Aufgaben automatisieren**



# 7

## Mustervergleich mit regulären Ausdrücken



Sicherlich sind Sie damit vertraut, zur Suche nach Text `Strg + F` zu drücken und dann die Wörter einzugeben, nach denen Sie Ausschau halten wollen. Mit *regulären Ausdrücken* können Sie jedoch noch einen Schritt weitergehen und ein *Textmuster* eingeben, nach dem gesucht werden soll. Beispielsweise werden Telefonnummern in den USA und Kanada in Form von drei Ziffern, einem Bindestrich und vier weiteren Ziffern angegeben (mit einer optionalen dreistelligen Bereichsvorwahl ganz am Anfang). Wenn Sie also so etwas wie 415-555-1234 sehen, können Sie zumindest erkennen, dass es sich um eine Telefonnummer handelt (auch wenn Sie nicht sagen können, dass es die richtige Nummer für den angegebenen Anschluss ist), während Ihnen klar ist, dass 4.155.551,234 keine Telefonnummer ist.

Reguläre Ausdrücke sind äußerst nützlich, aber viele Nicht-Programmierer wissen nicht viel darüber, obwohl moderne Textverarbeitungssysteme wie Microsoft Word und OpenOffice auch Such- und Ersetzfunktionen auf der Grundlage von regulären Ausdrücken anbieten. Dabei können reguläre Ausdrücke enorm viel Zeit sparen helfen, nicht nur bei der Benutzung von Software, sondern auch bei

der Programmierung. Der technische Autor Cory Doctorow hat sogar dafür plädiert, Anfängern zunächst reguläre Ausdrücke beizubringen und erst danach das eigentliche Programmieren:

*»Sich mit [regulären Ausdrücken] auszukennen, kann entscheiden, ob man ein Problem in drei Schritten lösen kann oder in 3000. Als Computerfreak ist Ihnen oft nicht mehr bewusst, dass die Probleme, die Sie mit wenigen Tastendrücken lösen, für andere Leute eine mühselige, fehleranfällige Quälerei darstellen.«<sup>1</sup>*

In diesem Kapitel schreiben Sie als Erstes ein Programm, das Textmuster *ohne* Zuhilfenahme regulärer Ausdrücke findet, um dann zu sehen, wie Sie die gleiche Aufgabe mithilfe regulärer Ausdrücke viel knapper fassen können. Ich zeige Ihnen die Grundlagen des Mustervergleichs mit regulären Ausdrücken und gehe dann zu einigen leistungsstarken Besonderheiten wie Stringersetzung und die Erstellung von eigenen Zeichenklassen über. Am Ende des Kapitels schreiben Sie ein Programm, das automatisch Telefonnummern und E-Mail-Adressen aus einem Textblock extrahieren kann.

## Textmuster ohne reguläre Ausdrücke finden

Nehmen wir an, Sie möchten eine (amerikanische) Telefonnummer in einem String finden. Das Muster haben Sie weiter vorn schon kennengelernt: drei Ziffern, ein Bindestrich, drei Ziffern, ein weiterer Bindestrich und schließlich vier Ziffern, beispielsweise 415-555-4242.

Schreiben wir nun die Funktion `isPhoneNumber()`, die prüft, ob ein String diesem Muster genügt, und dementsprechend entweder True oder False zurückgibt. Geben Sie dazu folgenden Code im Dateieditor ein und speichern Sie ihn als `isPhoneNumber.py`:

```
def isPhoneNumber(text):
    if len(text) != 12: ❶
        return False
    for i in range(0, 3):
        if not text[i].isdecimal(): ❷
            return False
    if text[3] != '-': ❸
        return False
    for i in range(4, 7):
        if not text[i].isdecimal(): ❹
```

---

<sup>1</sup> Cory Doctorow, »Here's what ICT should really teach kids: how to do regular expressions«, Guardian, 4. Dezember 2012, <http://www.theguardian.com/technology/2012/dec/04/ict-teach-kids-regular-expressions/>.

```
        return False
    if text[7] != '-': ❸
        return False
    for i in range(8, 12):
        if not text[i].isdecimal(): ❹
            return False
    return True ❺

print('415-555-4242 is a phone number:')
print(isPhoneNumber('415-555-4242'))
print('Moshi moshi is a phone number:')
print(isPhoneNumber('Moshi moshi'))
```

Die Ausgabe sieht wie folgt aus:

```
415-555-4242 is a phone number:
True
Moshi moshi is a phone number:
False
```

Der Code der Funktion `isPhoneNumber()` führt mehrere Prüfungen durch, um herauszufinden, ob der String in `text` eine gültige Telefonnummer ist. Wird nur einer dieser Tests nicht bestanden, gibt die Funktion `False` zurück. Als Erstes prüft der Code, ob der String genau zwölf Zeichen lang ist (❶). Danach prüft er, ob die Bereichsvorwahl (die ersten drei Zeichen des Textes) ausschließlich aus numerischen Zeichen besteht (❷). Der Rest der Funktion untersucht nacheinander die einzelnen Bestandteile des Musters für amerikanische Telefonnummern: Hinter der Bereichsvorwahl muss ein Bindestrich stehen (❸) und darauf müssen drei weitere numerische Zeichen (❹), ein weiterer Bindestrich (❺) und schließlich noch einmal vier Ziffern folgen (❻). Wenn alle Tests bestanden wurden, gibt das Programm `True` zurück (❼).

Der Aufruf von `isPhoneNumber()` mit dem Argument `'415-555-4242'` gibt daher `True` zurück, der Aufruf mit `'Moshi moshi'` dagegen `False`. Hier wird schon der erste Test nicht bestanden, da `'Moshi moshi'` nicht zwölf Zeichen lang ist.

Um dieses Textmuster in einem längeren String zu finden, müssen Sie noch mehr Code hinzufügen. Ersetzen Sie die letzten vier Aufrufe der Funktion `print()` in `isPhoneNumber.py` durch folgenden Code:

```
message = 'Call me at 415-555-1011 tomorrow. 415-555-9999 is my office.'
for i in range(len(message)):
    chunk = message[i:i+12] ❶
    if isPhoneNumber(chunk): ❷
        print('Phone number found: ' + chunk)
print('Done')
```

Die Ausgabe sieht wie folgt aus:

```
Phone number found: 415-555-1011
Phone number found: 415-555-9999
Done
```

Bei jedem Durchlauf durch die for-Schleife wird der Variablen chunk ein neuer Abschnitt von zwölf Zeichen aus message zugewiesen (❶). Beim ersten Durchlauf ist i gleich 0, weshalb message[0:12] zu chunk zugewiesen wird (also der String 'Call me at 4'). Bei der nächsten Iteration haben wir i gleich 1 und damit message[1:13] (der String 'all be at 41').

Dabei wird chunk jeweils der Funktion isPhoneNumber() übergeben, um herauszufinden, ob der Textabschnitt dem Muster einer Telefonnummer entspricht (❷). Wenn ja, wird der Abschnitt ausgegeben.

Wenn Sie die Schleife den gesamten Text von message durchlaufen lassen, stehen schließlich tatsächlich die zwölf Zeichen einer Telefonnummer in chunk. Die Schleife geht den gesamten String durch und prüft jeden einzelnen Abschnitt von zwölf Zeichen. Wenn sie dabei einen Abschnitt findet, der die Bedingungen von isPhoneNumber() erfüllt, gibt sie diesen aus. Nachdem message komplett abgearbeitet ist, geben wir Done aus.

In diesem Beispiel ist der String in message sehr kurz, er könnte aber auch Millionen von Zeichen lang sein. Das Programm würde auch in diesem Fall weniger als eine Sekunde benötigen, um ihn zu untersuchen. Ein vergleichbares Programm, das Telefonnummern mithilfe regulärer Ausdrücke findet, hat eine ähnliche Ausführungszeit, allerdings geht es viel schneller, das Programm zu schreiben.

## Textmuster mithilfe regulärer Ausdrücke finden

Das Telefonnummernsuchprogramm aus dem vorherigen Abschnitt funktioniert zwar, braucht aber eine ganze Menge Code, um eine eingeschränkte Aufgabe auszuführen: Die Funktion isPhoneNumber() umfasst 17 Zeichen und kann nur ein einziges Muster von Telefonnummern finden. Was aber, wenn jemand eine Telefonnummer in einem Format wie 415.555.4242 oder (415) 555-4242 schreibt? Oder wenn die Telefonnummer über eine Nebenstellenerweiterung verfügt, z. B. 415-555-4242 x99? Bei der Überprüfung solcher Werte würde isPhoneNumber() versagen. Natürlich könnten Sie noch mehr Code hinzufügen, um auch diese Muster abzudecken, doch es gibt eine einfachere Möglichkeit.

Reguläre Ausdrücke (englisch »regular expressions«, was oft zu »regex« abgekürzt wird) beschreiben Textmuster. Beispielsweise steht \d in einem regulären Ausdruck für eine Ziffer, also einen einzelnen numerischen Wert von 0 bis 9. Mit dem regulären Ausdruck \d\d\d-\d\d\d-\d\d\d können Sie in Python Text mit dem gleichen Aufbau finden, den Sie auch mit der Funktion `isPhoneNumber()` ermittelt haben: einen String aus drei Ziffern, einem Bindestrich, drei weiteren Ziffern, einem weiteren Bindestrich und schließlich vier Ziffern. Kein anderer String stimmt mit dem regulären Ausdruck \d\d\d-\d\d\d-\d\d\d überein.

Für reguläre Ausdrücke gibt es aber noch weitere und anspruchsvollere Möglichkeiten. Beispielsweise können Sie eine Zahl in geschweifte Klammern schreiben (z. B. `\{3\}`), um damit zu verlangen, dass das vorstehende Muster dreimal vorhanden sein muss. Das amerikanische Telefonnummernformat können Sie daher auch mithilfe des kürzeren regulären Ausdrucks `\d\{3\}-\d\{3\}-\d\{4\}` ausdrücken.

## Regex-Objekte erstellen

Alle Regex-Funktionen von Python befinden sich im Modul `re`. Um es zu importieren, geben Sie Folgendes in die interaktive Shell ein:

```
>>> import re
```

### Hinweis

Da Sie für die meisten Beispiele in diesem Kapitel das Modul `re` benötigen, müssen Sie es zu Anfang jedes Skripts bzw. bei jedem Neustart von IDLE importieren. Andernfalls erhalten Sie die Fehlermeldung `NameError: name 're' is not defined`.

Wenn Sie `re.compile()` einen Stringwert mit einem regulären Ausdruck übergeben, gibt die Funktion ein Regex-Musterobjekt zurück (oder kurz gesagt, ein Regex-Objekt).

Um ein Regex-Objekt für das Telefonnummernmuster zu erstellen, geben Sie in der interaktiven Shell Folgendes ein:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

Jetzt enthält die Variable `phoneNumRegex` ein Regex-Objekt.

### Rohstrings an re.compile() übergeben

Denken Sie daran, dass der Backslash (\) in Python als Maskierungszeichen verwendet wird. Der Stringwert '\n' ist daher kein umgekehrter Schrägstrich, auf den ein kleines *n* folgt, sondern ein einzelnes Zeichen, nämlich ein Zeilenumbruch. Um tatsächlich einen Backslash auszugeben, müssen Sie \\ schreiben. Der String '\\n' stellt also tatsächlich einen Backslash gefolgt von einem kleinen *n* dar. Wenn Sie vor das öffnende Anführungszeichen eines Strings allerdings ein r stellen, kennzeichnen Sie ihn als einen *Rohstring*, in dem die Backslashes als solche enthalten sind.

Da in regulären Ausdrücken häufig Backslashes vorkommen, ist es komfortabler, der Funktion `recompile()` einen Rohstring zu übergeben, anstatt alle Backslashes durch zusätzliche Backslashes zu maskieren. Ein String wie

`r'\d\d\d-\d\d\d-\d\d\d'` lässt sich viel einfacher eingeben als  
`'\\d\\d\\d-\\d\\d\\d-\\d\\d\\d'`.

### Vergleiche mit einem Regex-Objekt

Die Methode `search()` eines Regex-Objekts durchsucht den ihr übergebenen String nach Übereinstimmungen mit dem regulären Ausdruck. Sie gibt `None` zurück, wenn das Muster des Ausdrucks in dem String nicht zu finden ist, und anderenfalls ein Match-Objekt. Solche Objekte wiederum verfügen über die Methode `group()`, die den übereinstimmenden Text aus dem durchsuchten String zurückgibt. (Was es mit den namensgebenden Gruppen auf sich hat, erkläre ich in Kürze.) Geben Sie zum Ausprobieren Folgendes in die interaktive Shell ein:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())
Phone number found: 415-555-4242
```

Die Bezeichnung `mo` ist lediglich ein generischer Variablenname für Match-Objekte. Dieses Beispiel mag zunächst ziemlich kompliziert aussehen, allerdings können Sie schon auf einen Blick erkennen, dass es viel kürzer ist als *iPhoneNumber.py*, obwohl es genau die gleiche Aufgabe erledigt.

Hier übergeben wir das gesuchte Muster an `re.compile()` und speichern das resultierende Regex-Objekt in `phoneNumRegex`. Anschließend rufen wir die Methode `search()` für `phoneNumRegex` auf und übergeben ihr den String, den wir nach Übereinstimmungen durchsuchen wollen. Das Ergebnis der Suche wird in der Variablen `mo` gespeichert. In diesem Beispiel wissen wir bereits, dass das Muster in dem String

enthalten ist und dass daher ein Match-Objekt zurückgeben wird. Da `mo` also mit Sicherheit ein Match-Objekt enthält und nicht den Wert `None`, können wir `group()` für `mo` aufrufen, um den übereinstimmenden String zurückzugeben. Da wir `mo.group()` in einer `print`-Anweisung verwenden, wird der übereinstimmende Text (415-555-4242) ausgegeben.

### Zusammenfassung: Mustervergleich mit regulären Ausdrücken

Die Verwendung regulärer Ausdrücke in Python umfasst mehrere Schritte, die aber alle ziemlich einfach sind:

1. Importieren Sie das Regex-Modul mit `import re`.
2. Erstellen Sie mithilfe der Funktion `re.compile()` ein Regex-Objekt. (Verwenden Sie dazu am besten einen Rohstring.)
3. Übergeben Sie den String, den Sie durchsuchen wollen, an die Methode `search()` des Regex-Objekts. Sie gibt ein Match-Objekt zurück.
4. Rufen Sie die Methode `group()` des Match-Objekts auf, um einen String mit dem übereinstimmenden Text zurückzugeben.

#### Hinweis

Ich ermutige Sie zwar immer dazu, den Beispielcode in der interaktiven Shell auszuprobieren, aber Sie sollten sich auch einige der Testdienste für reguläre Ausdrücke im Internet ansehen, die Ihnen genau zeigen, ob ein regulärer Ausdruck mit einem von Ihnen eingegebenen Text übereinstimmt. Dazu empfehle ich Ihnen den Testdienst auf <http://regexpal.com/>.

## Weitere Möglichkeiten für den Mustervergleich mithilfe regulärer Ausdrücke

Nachdem Sie nun die grundlegenden Schritte kennen, um reguläre Ausdrücke in Python zu erstellen und für den Mustervergleich zu verwenden, wollen wir uns jetzt einige Besonderheiten ansehen, die die Möglichkeiten noch erweitern.

### Gruppierung durch Klammern

Nehmen wir an, Sie wollen die Bereichsvorwahl vom Rest der Telefon trennen. Mithilfe von Klammern können Sie dazu innerhalb des regulären Ausdrucks *Gruppen* anlegen: `(\d\d\d)-(\d\d\d-\d\d\d\d)`. Anschließend können Sie mit der Methode `group` des Match-Objekts gezielt den übereinstimmenden Text für eine einzelne Gruppe abrufen.

Der erste Satz von Klammern in einem Regex-String ist Gruppe 1, der zweite ist Gruppe 2. Durch die Übergabe des Integers 1 oder 2 an group() rufen Sie unterschiedliche Teile des übereinstimmenden Textes ab. Falls Sie 0 oder gar nichts übergeben, bekommen Sie den gesamten übereinstimmenden Text. Das können Sie wie folgt in der interaktiven Shell ausprobieren:

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

Wollen Sie alle Gruppen auf einmal abrufen, verwenden Sie die Methode groups() (mit der Pluralform des Namens!):

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

Da mo.groups() ein Tupel mit mehreren Werten zurückgibt, können Sie die Mehrfachzuweisung nutzen, um jeden dieser Werte einer eigenen Variable zuzuweisen, wie es im vorstehenden Code in der Zeile areaCode, mainNumber = mo.groups() geschieht.

Klammern haben in regulären Ausdrücken eine besondere Bedeutung. Was machen Sie aber, wenn Sie in Ihrem Text nach Klammern suchen müssen? Beispielsweise kann es sein, dass die Vorwahlnummer in Klammern steht. In diesem Fall müssen Sie die Zeichen für die öffnende und schließende Klammer mit einem Backslash maskieren. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> phoneNumRegex = re.compile(r'((\d\d\d)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My phone number is (415) 555-4242.')
>>> mo.group(1)
'(415)'
>>> mo.group(2)
'555-4242'
```

Die Maskierungssequenzen \(`` und ``\)` im Rohstring werden an `re.compile()` übergeben und dienen zur Suche nach Klammerzeichen im Text.

## Mithilfe der Pipe nach Übereinstimmungen mit mehreren Gruppen suchen

Der senkrechte Strich (|) wird als *Pipe* bezeichnet. Dieses Zeichen verwenden Sie, wenn Sie nach Übereinstimmungen mit einem von mehreren verschiedenen Ausdrücken suchen. Beispielsweise findet der reguläre Ausdruck `r'Batman|Tina Fey'` sowohl 'Batman' als auch 'Tina Fey'.

Wenn in dem durchsuchten String sowohl Batman als auch Tina Fey vorkommen, wird als Match-Objekt die erste dieser Übereinstimmungen zurückgegeben. Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> heroRegex = re.compile (r'Batman|Tina Fey')
>>> mo1 = heroRegex.search('Batman and Tina Fey.')
>>> mo1.group()
'Batman'

>>> mo2 = heroRegex.search('Tina Fey and Batman.')
>>> mo2.group()
'Tina Fey'
```

### Hinweis

Um *alle* Übereinstimmungen zu finden, verwenden Sie die Methode `findall()`, die im Abschnitt »Die Methode `findall()`« weiter hinten in diesem Kapitel beschrieben wird.

Mit der Pipe können Sie auch nach einem von mehreren verschiedenen Mustern als Teil des regulären Ausdrucks suchen. Nehmen wir an, Sie suchen nach Übereinstimmungen mit einem der Strings 'Batman', 'Batmobile', 'Batcopter' und 'Batbat'. Da alle diese Strings mit Bat beginnen, wäre es praktisch, dieses Präfix nur einmal angeben zu müssen. Mit Klammern und der Pipe ist das tatsächlich möglich:

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
>>> mo = batRegex.search('Batmobile lost a wheel')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'mobile'
```

Der Methodenaufruf `mo.group()` gibt den kompletten übereinstimmenden Text 'Batmobile' zurück, `mo.group(1)` dagegen nur den Text in der ersten Klammern-

gruppe, also 'mobile'. Mithilfe der Pipe und Klammern zur Gruppierung können Sie mehrere alternative Muster angeben, mit denen der reguläre Ausdruck übereinstimmen soll.

Wenn Sie nach dem Pipezeichen selbst suchen müssen, so maskieren Sie es mit einem Backslash (\|).

### Optionale Übereinstimmung mit dem Fragezeichen

Es kann vorkommen, dass ein Teil eines Suchmusters optional ist. In einem solchen Fall soll der reguläre Ausdruck eine Übereinstimmung unabhängig davon feststellen, ob dieser bestimmte Text vorhanden ist. Mit dem Zeichen ? markieren Sie die voranstehende Gruppe als optionalen Teil des Musters. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

Der Teil (wo)? in diesem regulären Ausdruck bedeutet, dass es sich bei wo um eine optionale Gruppe handelt. Der reguläre Ausdruck findet Texte, die null oder ein Vorkommen von wo enthalten, also sowohl 'Batwoman' als auch 'Batman'.

In unserem Telefonnummernbeispiel können wir damit nach Nummern mit und ohne Bereichsvorwahl suchen:

```
>>> phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')
>>> mo1 = phoneRegex.search('My number is 415-555-4242')
>>> mo1.group()
'415-555-4242'
>>> mo2 = phoneRegex.search('My number is 555-4242')
>>> mo2.group()
'555-4242'
```

Das Zeichen ? bedeutet also: »Suche nach null oder einem Vorkommen der Gruppe, die dem Fragezeichen vorausgeht.«

### Mit dem Sternchen nach null oder mehr Übereinstimmungen suchen

Das Sternchen \* bedeutet »suche nach null oder mehr Übereinstimmungen« der voranstehenden Gruppe. Diese Gruppe kann an der entsprechenden Stelle im Text

also beliebig oft vorkommen oder völlig fehlen. Sehen wir uns dazu noch einmal unser Batman-Beispiel an:

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'

>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

In dem Text mit 'Batman' gibt es null Vorkommen von wo in dem String, was durch (wo)\* aber genauso abgedeckt ist wie das einzelne Vorkommen in 'Batwoman' und die vier Vorkommen in 'Batwowowowoman'.

Wenn Sie nach dem Sternchen selbst suchen müssen, so maskieren Sie es mit einem Backslash (\\*).

### Mit dem Pluszeichen nach einer oder mehr Übereinstimmungen suchen

Mit dem Sternchen suchen Sie nach »null oder mehr Vorkommen«, mit dem Pluszeichen dagegen nach »ein oder mehr Vorkommen«. Hier muss die dem Pluszeichen vorausgehende Gruppe also *mindestens einmal* in dem String vorkommen; sie ist nicht optional. Geben Sie Folgendes in die interaktive Shell ein und vergleichen Sie das Ergebnis mit den Sternchen-Ausdrücken aus dem vorherigen Abschnitt:

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'

>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'

>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

Der reguläre Ausdruck Bat(wo)+man findet keine Übereinstimmung in 'The Adventures of Batman', da das Pluszeichen verlangt, dass der Ausdruck wo mindestens einmal vorkommt.

Wenn Sie nach dem Pluszeichen selbst suchen müssen, so maskieren Sie es mit einem Backslash (\+).

### Mit geschweiften Klammern nach einer genauen Zahl von Wiederholungen suchen

Wenn eine Gruppe in dem zu durchsuchenden Text eine bestimmte Anzahl von Malen hintereinander vorkommen soll, geben Sie in dem regulären Ausdruck hinter der Gruppe die gewünschte Anzahl in geschweiften Klammern an. Beispielsweise findet der Ausdruck (Ha){3} den String 'HaHaHa', aber nicht 'HaHa', da die Gruppe (Ha) in Letzterem nur zweimal vorkommt.

Anstatt einer festen Zahl können Sie auch einen Bereich angeben, indem Sie die Mindestanzahl, ein Komma und die Höchstanzahl in die geschweiften Klammern schreiben. So findet beispielsweise der reguläre Ausdruck (Ha){3,5} die Strings 'HaHaHa', 'HaHaHaHa' und 'HaHaHaHaHa'.

Wenn Sie nur ein Maximum oder ein Minimum angeben möchten, können Sie die erste bzw. zweite Zahl weglassen. Beispielsweise findet (Ha){3,} drei oder mehr Vorkommen der Gruppe (Ha), der Ausdruck (Ha){,5} dagegen null bis fünf Vorkommen. Mit geschweiften Klammern können Sie reguläre Ausdrücke kürzer fassen. Die beiden folgenden Ausdrücke stehen für das gleiche Muster:

```
(Ha){3}  
(Ha)(Ha)(Ha)
```

Noch deutlicher wird es anhand der beiden folgenden Ausdrücke, die ebenfalls beide mit demselben Muster übereinstimmen:

```
(Ha){3,5}  
((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))
```

Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> haRegex = re.compile(r'(Ha){3}')  
>>> mo1 = haRegex.search('HaHaHa')  
>>> mo1.group()  
'HaHaHa'  
  
>>> mo2 = haRegex.search('Ha')  
>>> mo2 == None  
True
```

Hier findet (Ha){3} zwar 'HaHaHa', aber nicht 'Ha'. Daher gibt search() den Wert None zurück.

## Gieriger und nicht gieriger Mustervergleich

Der Ausdruck `(Ha){3,5}` stimmt mit drei, vier und fünf Vorkommen von `Ha` in dem String `'HaHaHaHaHa'` überein. Wenn Sie diesen Mustervergleich durchführen, erhalten Sie jedoch tatsächlich `'HaHaHaHaHa'` zurück und keine der kürzeren Varianten, obwohl auch `'HaHaHaHa'` und `'HaHaHa'` gültige Übereinstimmungen sind. Warum ist das so?

Reguläre Ausdrücke sind in Python standardmäßig *gierig* (»greedy«), was bedeutet, dass in mehrdeutigen Situationen der längstmögliche String gefunden wird. Um die *nicht gierige* Version der geschweiften Klammern zu verwenden, die den kürzestmöglichen String zurückgibt, müssen Sie hinter die Klammern ein Fragezeichen stellen.

Geben Sie Folgendes in die interaktive Shell ein und beobachten Sie den Unterschied zwischen der gierigen und der nicht gierigen Version der geschweiften Klammern:

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?)'
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

Das Fragezeichen hat in regulären Ausdrücken also zwei Bedeutungen: Es kann eine nicht gierige Suche verlangen, aber auch eine Gruppe als optional kennzeichnen. Diese beiden Bedeutungen sind völlig unabhängig voneinander.

## Die Methode `findall()`

Neben der Methode `search()` verfügen Regex-Objekte auch über die Methode `findall()`. Wie das folgende Beispiel in der interaktiven Shell zeigt, gibt `search()` nur ein Match-Objekt für die *erste* Übereinstimmung in dem durchsuchten String zurück:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d\d\d\d\d\d')
>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

Dagegen gibt `findall()` kein Match-Objekt zurück, sondern eine Liste der Strings aller Übereinstimmungen, die in dem durchsuchten Text zu finden sind – *sofern*

es keine Gruppen in dem regulären Ausdruck gibt. Das können Sie an dem folgenden Beispiel in der interaktiven Shell erkennen:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d') # Ohne Gruppen
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000']
```

Enthält der reguläre Ausdruck jedoch Gruppen, so gibt `.findall()` eine Liste von Tupeln zurück, die jeweils für eine gefundene Übereinstimmung stehen. Die Elemente dieser Tupel sind die übereinstimmenden Strings für die verschiedenen Gruppen in dem regulären Ausdruck. Um sich das anzusehen, geben Sie Folgendes in die interaktive Shell ein:

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # Mit Gruppen
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

Zusammenfassend gesagt, verhält sich `.findall()` also wie folgt:

1. Wird die Methode `.findall()` auf reguläre Ausdrücke ohne Gruppen angewendet, z. B. `\d\d\d-\d\d\d-\d\d\d\d`, so gibt sie eine Liste der übereinstimmenden Strings zurück, z. B. `['415-555-9999', '212-555-0000']`.
2. Wird die Methode `.findall()` auf reguläre Ausdrücke mit Gruppen angewendet, z. B. `(\d\d\d)-(\d\d\d)-(\d\d\d\d)`, so gibt sie eine Liste von Tupeln aus Strings zurück (mit je einem String für jede Gruppe), z. B. `[('415', '555', '1122'), ('212', '555', '0000')]`.

## Zeichenklassen

In dem Telefonnummernbeispiel haben Sie gelernt, dass `\d` für eine beliebige Ziffer stehen kann. Das bedeutet, dass `\d` eine Abkürzung für den regulären Ausdruck `(0|1|2|3|4|5|6|7|8|9)` ist. Tabelle 7–1 zeigt noch weitere solcher *Kurzschreibweisen für Zeichenklassen*.

Abkürzung	Bedeutung
<code>\d</code>	Eine beliebige Ziffer von 0 bis 9. (Die Bezeichnung <i>d</i> stammt vom englischen Wort <i>digit</i> für Ziffer.)
<code>\D</code>	Ein beliebiges Zeichen mit Ausnahme der Ziffern von 0 bis 9.
<code>\w</code>	Ein beliebiger Buchstabe, eine beliebige Ziffer oder ein Unterstrich. (Das <i>w</i> steht für <i>word</i> .)
<code>\W</code>	Ein beliebiges Zeichen mit Ausnahme von Buchstaben, Ziffern und dem Unterstrich.

Abkürzung	Bedeutung
\s	Ein Leerzeichen, Tabulator oder Zeichenumbruch. (Das s steht für <i>space</i> , also »Leerzeichen«.)
\S	Ein beliebiges Zeichen mit Ausnahme des Leerzeichens, des Tabulators und des Zeichenumbruchs.

**Tab. 7-1** Abkürzungen für gebräuchliche Zeichenklassen

Zeichenklassen sind praktisch, um reguläre Ausdrücke kürzer zu fassen. Mit der Zeichenklasse [0-5] finden Sie nur Ziffern von 0 bis 5. Dies ist kürzer als (0|1|2|3|4|5).

Zur Veranschaulichung probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 drummers, 11 pipers, 10 lords, 9 ladies, 8 maids, 7
swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1 partridge')
['12 drummers', '11 pipers', '10 lords', '9 ladies', '8 maids', '7 swans', '6
geese', '5 rings', '4 birds', '3 hens', '2 doves', '1 partridge']
```

Der reguläre Ausdruck \d+\s\w+ findet Text mit ein oder mehreren Ziffern am Anfang (\d+), auf die ein Weißraumzeichen (\s) und daraufhin ein oder mehrere Buchstaben, Ziffern oder Unterstriche folgen (\w+). Die Methode `.findall()` gibt alle Strings, die mit diesem Muster übereinstimmen, in Form einer Liste zurück.

## Eigene Zeichenklassen bilden

Wenn Sie nach Übereinstimmungen mit einer bestimmten Auswahl von Zeichen suchen, sind die vorhandenen Zeichenklassen (\d, \w, \s usw.) manchmal zu weit gefasst. Mithilfe von eckigen Klammern können Sie jedoch eigene Zeichenklassen definieren. Beispielsweise findet die Klasse [aeiouAEIOU] alle Vokale, sowohl in Klein- als auch in Großschreibung. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> vowelRegex = re.compile(r'[aeiouAEIOU]')
>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')
['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']
```

Zahlen- und Buchstabebereiche können Sie dabei auch mithilfe von Bindestrichen angeben. Beispielsweise steht die Zeichenklasse [a-zA-Z0-9] für alle Klein- und Großbuchstaben sowie Ziffern.

Innerhalb eckiger Klammern werden die üblichen Symbole für reguläre Ausdrücke nicht als Symbole aufgefasst, sondern als das betreffende Zeichen. Das bedeutet, dass Sie Zeichen wie ., \*, ? und () nicht mit einem Backslash maskieren

müssen. Die Zeichenklasse [0-5.] steht für die Ziffern 0 bis 5 und den Punkt; in diesem Fall schreiben Sie also *nicht* [0-5\.].

Wenn Sie einen Zirkumflex (^) hinter die öffnende eckige Klasse setzen, definieren Sie eine *negative Zeichenklasse*. Sie stimmt mit allen Zeichen überein, die *nicht* angegeben werden. Geben Sie zur Veranschaulichung Folgendes in die interaktive Shell ein:

```
>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')
>>> consonantRegex.findall('RoboCop eats baby food. BABY FOOD.')
['R', 'b', 'c', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f', 'd', '.', ' ', 'B', 'B', 'Y', ' ', 'F', 'D', '.']
```

Diese Zeichenklasse steht nicht für alle Vokale, sondern für alle Zeichen, die keine Vokale sind.

## Zirkumflex und Dollarzeichen

Wenn Sie den Zirkumflex (^) an den Anfang eines regulären Ausdrucks setzen, verlangen Sie damit, dass die Übereinstimmung am *Anfang* des durchsuchten Texts vorhanden sein muss. Wenn der gesuchte String dagegen mit dem Muster aus dem regulären Ausdruck *enden* soll, hängen Sie an den Ausdruck ein Dollarzeichen an. Sie können ^ und \$ auch zusammen verwenden, um eine Übereinstimmung mit dem gesamten String zu verlangen und nicht nur mit einem Teilstring.

Beispielsweise findet der reguläre Ausdruck r'^Hello' alle Strings, die mit 'Hello' beginnen:

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello world!')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said hello.') == None
True
```

Der reguläre Ausdruck r'\d\$' findet Strings, die mit einer Ziffer enden:

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<_sre.SRE_Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('Your number is forty two.') == None
True
```

Der reguläre Ausdruck r'^\d+\\$' findet Strings, die mit einer oder mehreren Ziffern beginnen und enden:

```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<sre.SRE_Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12 34567890') == None
True
```

Die letzten beiden Aufrufe von `search()` zeigen, wie Sie `^` und `$` kombiniert einsetzen können, um zu verlangen, dass der gesamte String mit dem regulären Ausdruck übereinstimmen muss.

## Das Jokerzeichen

Der Punkt gilt in einem regulären Ausdruck als *Jokerzeichen* und steht für jedes beliebige Zeichen mit Ausnahme eines Zeilenumbruchs:

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

Beachten Sie, dass der Punkt nur für ein einzelnes Zeichen steht, weshalb bei dem Wort `flat` im vorstehenden Beispiel nur die Übereinstimmung `lat` gefunden wird. Um einen Punkt als solchen in einem Text zu finden, müssen Sie das Zeichen mit einem Backslash maskieren (`\.`).

## Beliebige Übereinstimmungen mit Punkt-Stern finden

Manchmal müssen Sie Übereinstimmungen mit beliebigem Text finden können. Nehmen wir an, Sie suchen eine Zeichenfolge, die aus dem String '`First Name:`', einem beliebigen Namen, dem String '`Last Name:`' und einem weiteren beliebigen Namen besteht. Dazu können Sie die Zeichenkombination Punkt-Stern verwenden (`.*`); sie steht praktisch für »alles Mögliche«. Das setzt sich aus der Bedeutung des Punkts (»jedes beliebige einzelne Zeichen mit Ausnahme eines Zeilenumbruchs«) und des Sterns (»null oder mehr Vorkommen des vorstehenden Zeichens«) zusammen.

Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

Die Punkt-Stern-Kombination ist gierig und versucht immer so viel Text zu finden wie möglich. Um beliebigen Text auf nicht gierige Weise zu finden, müssen Sie eine Kombination aus Punkt, Stern und Fragezeichen verwenden (`.*?`). Wie bei den geschweiften Klammern weist das Fragezeichen Python auch hier an, eine nicht gierige Suche durchzuführen.

Um sich den Unterschied zwischen der gierigen und der nicht gierigen Suche anzusehen, probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'

>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

Beide regulären Ausdrücke bedeuten grob gesagt: »Suche eine öffnende spitze Klammer, auf die beliebiger Text und eine schließende spitze Klammer folgen.« Im Text '`<To serve man> for dinner.>`' gibt es aber zwei schließende spitze Klammern. In der nicht gierigen Version findet Python den kürzestmöglichen String, '`<To server man>`', in der gierigen Version dagegen den längsten, '`<To serve man> for dinner.>`'.

## Zeilenumbrüche mit dem Punktsymbol finden

Die Kombination Punkt-Stern steht für beliebige Zeichen außer einem Zeilenumbruch. Wenn Sie `re.DOTALL` als zweites Argument an `re.compile()` übergeben, finden Sie mit dem Punkt jedoch Übereinstimmungen mit *sämtlichen* Zeichen, auch mit einem Zeilenumbruch:

```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.'

>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

Das Regex-Objekt `noNewlineRegex` wird erstellt, ohne dass `re.DOTALL` an `re.compile()` übergeben wird, weshalb es nur den Text bis zum ersten Zeilenumbruch findet.

Dagegen findet `newlineRegex`, bei dem `re.DOTALL` übergeben wurde, den gesamten Text des Strings einschließlich der Zeilenumbrüche.

## Übersicht über Regex-Symbole

In diesem Kapitel spielen Schreibweisen und Symbole eine große Rolle. Die folgende Aufstellung gibt einen Überblick über die Notation für reguläre Ausdrücke, die Sie bis jetzt gelernt haben:

- ? findet null oder ein Vorkommen der vorausgehenden Gruppe.
- \* findet null oder mehr Vorkommen der vorausgehenden Gruppe.
- + findet ein oder mehr Vorkommen der vorausgehenden Gruppe.
- {n} findet genau *n* Vorkommen der vorausgehenden Gruppe.
- {n,} findet *n* oder mehr Vorkommen der vorausgehenden Gruppe.
- {,m} findet 0 bis *m* Vorkommen der vorausgehenden Gruppe.
- {n,m} findet *n* bis *m* Vorkommen der vorausgehenden Gruppe.
- {n,m}? , \*? und +? führen eine nicht gierige Suche nach der vorausgehenden Gruppe durch.
- ^spam bedeutet, dass der String mit *spam* beginnen muss.
- spa\$ bedeutet, dass der String mit *spam* enden muss.
- . steht für ein beliebiges Zeichen mit Ausnahme eines Zeilenumbruchs.
- \d, \w und \s stehen für eine beliebige Ziffer, ein beliebiges Wortzeichen (Buchstaben, Ziffer oder Unterstrich) bzw. ein beliebiges Weißraumzeichen.
- \D, \W und \S stehen für ein beliebiges Zeichen, das keine Ziffer, kein Wortzeichen (Buchstaben, Ziffer oder Unterstrich) bzw. kein Weißraumzeichen ist.
- [abc] findet eines der in Klammern angegebenen Zeichen (hier also *a*, *b* oder *c*).
- [^abc] findet jedes Zeichen außer denen, die in den Klammern angegeben sind.

## Übereinstimmungen ohne Berücksichtigung der Groß- und Kleinschreibung

Normalerweise stimmen reguläre Ausdrücke nur mit Text überein, der die gleiche Groß- und Kleinschreibung aufweist. So finden beispielsweise die folgenden Ausdrücke jeweils völlig unterschiedliche Strings:

```
>>> regex1 = re.compile('RoboCop')
>>> regex2 = re.compile('ROBOCOP')
>>> regex3 = re.compile('rob0cop')
>>> regex4 = re.compile('RobocOp')
```

Es kann aber durchaus Fälle geben, in denen es auf die Groß- und Kleinschreibung gar nicht ankommt. Um die Unterscheidung bei regulären Ausdrücken auszuschalten, übergeben Sie `re.IGNORECASES` oder kurz `re.I` als zweites Argument an `re.compile()`:

```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop is part man, part machine, all cop.').group()
'RoboCop'

>>> robocop.search('ROBOCOP protects the innocent.').group()
'ROBOCOP'

>>> robocop.search('AI, why does your programming book talk about robocop so
much?').group()
'robocop'
```

## Strings mit der Methode `sub()` ersetzen

Mit regulären Ausdrücken können Sie nicht nur Text finden, der einem bestimmten Muster folgt, sondern ihn auch durch anderen Text ersetzen. Die Methode `sub()` (für *substitute*, also »ersetzen«) von Regex-Objekten nimmt zwei Argumente entgegen. Das erste ist der String, der jegliche gefundenen Übereinstimmungen ersetzen soll, das zweite der String des regulären Ausdrucks. Der Rückgabewert dieser Methode ist der String mit dem ersetzenen Text.

Zur Veranschaulichung probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to
Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

Es kann jedoch auch vorkommen, dass Sie Teile des übereinstimmenden Textes in der Ersetzung verwenden wollen. Im ersten Argument von `sub()` können Sie Angaben wie `\1`, `\2`, `\3` usw. machen, was bedeutet, dass der Text der Gruppe 1, 2, 3 usw. an der betreffenden Stelle in den neuen Text aufgenommen werden soll.

Nehmen wir an, Sie wollen die Namen der Geheimagenten nicht komplett zensieren, sondern nur auf ihren Anfangsbuchstaben reduzieren. Dazu verwenden Sie den regulären Ausdruck `Agent (\w)\w*` und übergeben `r'\1****'` als erstes Argument an `sub()`. Die `\1` in diesem String wird durch den Text ersetzt, der mit Gruppe 1 übereinstimmt, also mit der Gruppe `(\w)` des regulären Ausdrucks.

```
>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent Carol that Agent
Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

## Umgang mit komplizierten regulären Ausdrücken

Reguläre Ausdrücke sind eine feine Sache, vor allem, wenn das gesuchte Textmuster einfach ist. Allerdings kann es sehr lange und verwickelte reguläre Ausdrücke erfordern, um komplexe Textmuster zu finden. Ein wenig entschärfen können Sie eine solche Situation, indem Sie den String mit dem regulären Ausdruck übersichtlich gliedern und mit Kommentaren versehen. Dazu müssen Sie die Funktion `re.compile()` jedoch anwisen, Weißraumzeichen und Kommentare in dem String zu ignorieren. Das tun Sie, indem Sie die Variable `re.VERBOSE` als zweites Argument an `re.compile()` übergeben.

Nehmen wir an, Sie haben einen regulären Ausdruck wie den folgenden, der nun wirklich nicht leicht zu verstehen ist:

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?)?(\s|-|\.)?\d{3}(\s|-|\.)\d{4}
(\s*(ext|x|ext.)\s*\d{2,5})?')'
```

Im »ausführlichen Modus« (*verbose*) können Sie ihn nun wie folgt auf mehrere Zeilen verteilen und mit Kommentaren versehen:

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?                      # Bereichsvorwahl
    (\s|-|\.)?                            # Trennzeichen
    \d{3}                                # Erste 3 Stellen
    (\s|-|\.)                            # Trennzeichen
    \d{4}                                # Letzte 4 Stellen
    (\s*(ext|x|ext.)\s*\d{2,5})?          # Durchwahl
)''', re.VERBOSE)
```

Hier dient die Schreibweise mit dreifachen Anführungszeichen ('''') dazu, einen mehrzeiligen String zu erstellen, sodass Sie die Definition des regulären Ausdrucks auf mehrere Zeilen verteilen und damit übersichtlicher gestalten können.

Innerhalb von Strings für reguläre Ausdrücke gelten die gleichen Regeln für Kommentare wie im normalen Python-Code: Das Zeichen `#` und alle darauf folgenden Zeichen bis zum Ende der Zeile werden ignoriert. Auch die zusätzlichen Leerzeichen in dem mehrzeiligen String werden nicht als Bestandteile des zu findenden Textmusters gewertet. Dadurch können Sie reguläre Ausdrücke auf eine Weise schreiben, die sie leichter lesbar macht.

## Die Variablen `re.IGNORECASE`, `re.DOTALL` und `re.VERBOSE` kombinieren

Was aber, wenn Sie sowohl `re.VERBOSE` verwenden wollen, um Kommentare in einen regulären Ausdruck aufzunehmen, als auch `re.IGNORECASE`, um die Groß- und Kleinschreibung zu ignorieren? Leider nimmt `re.compile()` nur einen einzigen Wert als zweites Argument entgegen. Diese Einschränkung können Sie jedoch umgehen, indem Sie die Variablen `re.IGNORECASE`, `re.DOTALL` und `re.VERBOSE` mithilfe der Pipe (`|`) kombinieren, die in diesem Zusammenhang als *bitweiser OR-Operator* fungiert.

Wenn ein regulärer Ausdruck also nicht zwischen Groß- und Kleinschreibung unterscheiden, mit dem Punktsymbol aber auch Zeilenumbrüche finden soll, müssen Sie den Aufruf von `re.compile()` wie folgt schreiben:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL)
```

Wenn Sie im zweiten Argument alle drei Optionen angeben möchten, sieht das folgendermaßen aus:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL | re.VERBOSE)
```

Diese Syntax ist etwas almodisch und stammt noch aus früheren Versionen von Python. Bitweise Operatoren zu beschreiben, würde den Rahmen dieses Buchs sprengen, allerdings können Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) Quellen für weitere Informationen finden. Es gibt auch noch weitere Optionen, die Sie als zweites Argument übergeben können. Sie sind nicht sehr gebräuchlich, werden in den angeführten Quellen aber ebenfalls beschrieben.

## Projekt: Extraktionsprogramm für Telefonnummern und E-Mail-Adressen

Stellen Sie sich vor, jemand hat Ihnen die langweilige Aufgabe aufs Auge gedrückt, sämtliche Telefonnummern und E-Mail-Adressen auf einer umfangreichen Webseite oder in einem langen Dokument zu finden. Wenn Sie den Text manuell durchforsten, werden Sie dazu sicherlich lange Zeit benötigen. Haben Sie dagegen ein Programm zur Hand, das den Inhalt der Zwischenablage nach Telefonnummern und E-Mail-Adressen durchsucht, können Sie einfach den vorliegenden Text mit **[Strg] + [A]** komplett markieren und mit **[Strg] + [C]** in die Zwischenablage kopieren und dieses Programm ausführen, das dann den Text in der Zwischenablage durch eine Liste der gefundenen Telefonnummern und E-Mail-Adressen ersetzt.

Wenn Sie ein neues Projekt angehen, ist die Versuchung groß, gleich damit anzufangen, Code zu schreiben. Meistens ist es jedoch besser, sich zunächst einmal zurückzulehnen und das Gesamtbild zu betrachten. Ich empfehle Ihnen, als Erstes eine Übersicht darüber anzufertigen, was das Programm tun soll. Denken Sie dabei noch nicht an den Code – darum werden Sie sich später kümmern. Halten Sie sich zunächst einmal an die groben Züge.

Das Extraktionsprogramm für Telefonnummern und E-Mail-Adressen in unserem Beispiel muss Folgendes tun:

- Den Text aus der Zwischenablage beziehen
- Alle Telefonnummern und E-Mail-Adressen in dem Text finden
- Die gefundenen Nummern und Adressen in die Zwischenablage kopieren

Als Nächstes können Sie überlegen, wie Sie das im Code umsetzen können. Der Code muss Folgendes tun:

- Strings mithilfe des Moduls `pyperclip` kopieren und einfügen
- Zwei reguläre Ausdrücke für die Suche nach Telefonnummern und nach E-Mail-Adressen erstellen
- Sämtliche Übereinstimmungen mit den regulären Ausdrücken finden, nicht nur die jeweils erste
- Die gefundenen Strings übersichtlich formatieren und zu einem einzelnen String kombinieren, der eingefügt werden kann
- Eine Meldung anzeigen, wenn keine Übereinstimmungen gefunden werden konnten

Diese Liste ist praktisch der Fahrplan für Ihr Projekt. Wenn Sie den Code schreiben, konzentrieren Sie sich immer nur auf einen dieser Schritte auf einmal. Jede dieser Teilaufgaben lässt sich gut handhaben und in Python mithilfe von Konstruktionen ausdrücken, die Sie bereits kennen.

### Schritt 1: Einen regulären Ausdruck für Telefonnummern erstellen

Als Erstes müssen Sie einen regulären Ausdruck für die Suche nach Telefonnummern erstellen. Legen Sie dazu eine neue Datei an, geben Sie den folgenden Code ein und speichern Sie sie als `phoneAndEmail.py`:

```
#! python3
# phoneAndEmail.py - Findet Telefonnummern und E-Mail-Adressen
# in der Zwischenablage.

import pyperclip, re
```

```

phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?
    (\s|-|.)?
    \d{3}
    (\s|-|.)
    \d{4}
    (\s*(ext|x|ext.)\s*\d{2,5})? # Durchwahl
)''', re.VERBOSE)

# TODO: Regex für E-Mail-Adressen erstellen.

# TODO: Übereinstimmungen im Text in der Zwischenablage finden.

# TODO: Ergebnisse in die Zwischenablage kopieren.

```

Die TODO-Kommentare bilden das Skelett des Programms. Sie werden sie ersetzen, wenn Sie den betreffenden Code schreiben.

Da (amerikanische) Telefonnummern mit einer optionalen Bereichsvorwahl beginnen, steht hinter der Gruppe für diese Vorwahl ein Fragezeichen. Der Bereich kann einfach durch drei Ziffern angegeben sein (also durch `\d{3}`), aber auch durch drei Ziffern in Klammern `(\(\d{3}\))`. Daher müssen Sie diese beiden Varianten durch eine Pipe kombinieren. Um später noch zu wissen, wofür `(\d{3}|\(\d{3}\))?` steht, können Sie in diesem Teil des mehrzeiligen Strings den Kommentar `# Bereichsvorwahl` angeben.

Bei dem Trennzeichen für die einzelnen Bestandteile einer Telefonnummer kann es sich um ein Leerzeichen (`\s`), einen Bindestrich oder einen Punkt handeln, weshalb Sie auch diese drei Varianten durch Pipes kombinieren. Die folgenden Teile des regulären Ausdrucks sind unmittelbar einsichtig: drei Ziffern, gefolgt von einem weiteren Trennzeichen, gefolgt von vier Ziffern. Der letzte Teil ist die optionale Erweiterung, die aus einer beliebigen Anzahl von Leerzeichen gefolgt von ext, x oder ext. und zwei bis fünf weiteren Ziffern besteht.

## Schritt 2: Einen regulären Ausdruck für E-Mail-Adressen erstellen

Da Sie auch einen regulären Ausdruck für E-Mail-Adressen benötigen, ergänzen Sie Ihr Programm wie folgt:

```

#!/usr/bin/python3
# phoneAndEmail.py - Findet Telefonnummern und E-Mail-Adressen
# in der Zwischenablage.

import pyperclip, re

phoneRegex = re.compile(r'''(
-- schnipp --

```

```
# Regulärer Ausdruck für E-Mail-Adressen
emailRegex = re.compile(r'''(
    [a-zA-Z0-9._%+-]+ # Benutzername ❶
    @                 # @-Symbol ❷
    [a-zA-Z0-9.-]+   # Domänenname ❸
    (\.[a-zA-Z]{2,4}) # Punkt + irgendetwas
)''', re.VERBOSE)

# TODO: Übereinstimmungen im Text in der Zwischenablage finden.

# TODO: Ergebnisse in die Zwischenablage kopieren.
```

Der Teil der E-Mail-Adresse, der den Benutzernamen angibt, besteht aus einem oder mehr Zeichen, bei denen es sich um Klein- und Großbuchstaben, Ziffern, Punkte, Unterstriche, Prozentzeichen, Pluszeichen und Bindestriche handeln kann. Die gesamten Auswahlmöglichkeiten können Sie in die Zeichenklasse [a-zA-Z0-9.\_%+-] aufnehmen (❶).

Benutzer- und Domänenname sind durch das Zeichen @ getrennt (❷). Beim Domänennamen sind weniger Zeichen zulässig, nämlich nur Buchstaben, Ziffern, Punkte und Unterstriche, also [a-zA-Z0-9.-] (❸). Als Letztes folgt noch die *Top-Level-Domäne*, also der Punkt gefolgt von einer beliebigen Buchstabenfolge von zwei bis vier Zeichen Länge.

Für das Format von E-Mail-Adressen gibt es eine Menge eigenartiger Regeln. Mit dem hier angegebenen regulären Ausdruck können Sie nicht alle möglichen gültigen E-Mail-Adressen finden, aber praktisch alle, denen Sie in der Praxis begegnen werden.

### Schritt 3: Alle Überstimmungen im Inhalt der Zwischenablage finden

Nachdem Sie die regulären Ausdrücke für Telefonnummern und E-Mail-Adressen aufgestellt haben, können Sie die eigentliche Arbeit, den Text in der Zwischenablage nach sämtlichen Übereinstimmungen zu durchsuchen, dem Python-Modul `re` überlassen. Die Funktion `pyperclip.paste()` ruft den Stringwert des Textes in der Zwischenablage ab und die Regex-Methode `.findall` gibt eine Liste der gefundenen Tupel zurück.

Das Programm sieht jetzt wie folgt aus:

```
#! python3
# phoneAndEmail.py - Findet Telefonnummern und E-Mail-Adressen
# in der Zwischenablage.

import pyperclip, re

phoneRegex = re.compile(r'''(
```

```
-- schnipp --

# Findet Übereinstimmungen im Text in der Zwischenablage.
text = str(pyperclip.paste())
matches = [] ❶
for groups in phoneRegex.findall(text): ❷
    phoneNum = '-' .join([groups[1], groups[3], groups[5]])
    if groups[8] != '':
        phoneNum += ' x' + groups[8]
    matches.append(phoneNum)
for groups in emailRegex.findall(text): ❸
    matches.append(groups[0])

# TODO: Ergebnisse in die Zwischenablage kopieren.
```

Es gibt für jede Übereinstimmung ein Tupel mit Strings für die einzelnen Gruppen in dem regulären Ausdruck. Denken Sie daran, dass Gruppe 0 für den gesamten Ausdruck steht und das Tupel am Index 0 daher das ist, an dem Sie interessiert sind.

Wie Sie in (❶) sehen, werden die Übereinstimmungen in der Listenvariablen `matches` gespeichert. Sie ist zu Anfang leer, wird aber durch zwei `for`-Schleifen gefüllt. Für die E-Mail-Adressen hängen Sie einfach die Gruppe 0 jeder Übereinstimmung an (❸). Bei den Telefonnummern ist das jedoch nicht das, was wir wollen. Das Programm kann zwar Telefonnummern in verschiedenen Formaten erkennen, doch im Ergebnis sollen sie in einem einzigen Standardformat vorliegen. Daher wird der String in der Variable `phoneNum` aus den Gruppen 1, 3, 5 und 8 des gefundenen Textes aufgebaut (❷). (Bei diesen Gruppen handelt es sich um die Bereichsvorwahl, die ersten drei Stellen, die letzten vier Stellen und die Erweiterung.)

#### Schritt 4: Die gefundenen Übereinstimmungen zu einem String kombinieren

Die E-Mail-Adressen und Telefonnummern liegen jetzt als Liste von Strings in `matches` vor und warten darauf, dass Sie sie wieder in die Zwischenablage kopieren. Allerdings nimmt die Funktion `pyperclip.copy()` keine Stringlisten entgegen, sondern nur einzelne Stringwerte. Daher müssen Sie zunächst die Methode `join()` für `matches` aufrufen.

Um deutlicher sehen zu können, wie das Programm funktioniert, geben wir alle gefundenen Übereinstimmungen im Terminal aus. Wenn das Programm keine Telefonnummern oder E-Mail-Adressen findet, soll es eine entsprechende Meldung anzeigen.

Ergänzen Sie das Programm wie folgt:

```
#! python3
# phoneAndEmail.py - Findet Telefonnummern und E-Mail-Adressen
# in der Zwischenablage.

-- schnipp --
for groups in emailRegex.findall(text):
    matches.append(groups[0])

# Kopiert die Ergebnisse in die Zwischenablage
if len(matches) > 0:
    pyperclip.copy('\n'.join(matches))
    print('Copied to clipboard:')
    print('\n'.join(matches))
else:
    print('No phone numbers or email addresses found.')
```

## Das Programm ausführen

Für einen Beispieldurchlauf öffnen Sie in Ihrem Webbrowser die Kontaktseite von No Starch Press auf <http://www.nostarch.com/contactus.htm>. Drücken Sie [Strg] + [A], um den gesamten Text auf der Seite zu kopieren, und fügen Sie ihn dann mit [Strg] + [C] in die Zwischenablage ein. Wenn Sie nun das Programm ausführen, erhalten Sie folgende Ausgabe:

```
Copied to clipboard:
800-420-7240
415-863-9900
415-863-9950
info@nostarch.com
media@nostarch.com
academic@nostarch.com
help@nostarch.com
```

## Ideen für ähnliche Programme

Es gibt viele mögliche Anwendungen dafür, Textmuster zu erkennen und eventuell mit `sub()` zu ersetzen:

- Website-URLs finden, die mit `http://` oder `https://` beginnen
- Kalenderdaten in unterschiedlichen Formaten (wie `3/1q4/2015`, `03-14-2015` oder `2015/3/14`) durch Angaben in einem einheitlichen Format ersetzen
- Sensible Informationen wie Sozialversicherungs- oder Kreditkartennummern entfernen
- Übliche Fehler wie mehrere Leerzeichen zwischen Wörtern, versehentliche versehentliche Wortwiederholungen oder mehrere Ausrufezeichen am Satzende aufspüren. Die sind echt nervig!!!

## Zusammenfassung

Ein Computer kann Texte sehr schnell durchsuchen, allerdings müssen Sie ihm genau sagen, wonach er Ausschau halten soll. Mithilfe von regulären Ausdrücken können Sie die Zeichenmuster angeben, nach denen Sie suchen. Manche Textverarbeitungs- und Tabellenkalkulationsprogramme bieten Suchen- und Ersetzenfunktionen, bei denen Sie reguläre Ausdrücke verwenden können.

Mit dem im Lieferumfang von Python enthaltenen Modul `re` können Sie Regex-Objekte erstellen. Diese Objekte verfügen über verschiedene Methoden, etwa `search()`, um eine einzelne Übereinstimmung zu finden, `.findall()`, um alle Übereinstimmungen zu finden, und `sub()`, um den gefundenen Text durch anderen Text zu ersetzen.

Zur Syntax von regulären Ausdrücken gibt es noch mehr zu sagen als das, was in diesem Kapitel steht. Weitere Informationen erhalten Sie in der offiziellen Python-Dokumentation auf <http://docs.python.org/3/library/re.html>. Auch die Tutorial-Website <http://www.regular-expressions.info/> ist eine wertvolle Quelle.

Nachdem Sie jetzt wissen, wie Sie Strings bearbeiten und suchen, erfahren Sie als Nächstes, wie Sie in Dateien auf der Festplatte des Computers lesen und schreiben.

## Wiederholungsfragen

1. Welche Funktion erstellt Regex-Objekte?
2. Warum werden beim Erstellen von Regex-Objekten häufig Rohstrings verwendet?
3. Was gibt die Methode `search()` zurück?
4. Wie können Sie die Strings der übereinstimmenden Texte aus dem Match-Objekt gewinnen?
5. Wofür steht die Gruppe 0 bei dem regulären Ausdruck `r'(\d\d\d)-(\d\d\d-\d\d\d\d\d)'`? Wofür stehen Gruppe 1 und Gruppe 2?
6. In regulären Ausdrücken haben Klammern und Punkte eine besondere Bedeutung. Wie suchen Sie mit einem regulären Ausdruck nach Klammern und Punkten im Text?
7. Wann gibt die Methode `.findall()` eine Liste von Strings zurück und wann eine Liste von Tupeln?
8. Was bedeutet das Zeichen | in einem regulären Ausdruck?
9. Welche zwei Bedeutungen kann das Zeichen ? in einem regulären Ausdruck haben?
10. Was ist der Unterschied zwischen den Zeichen + und \* in einem regulären Ausdruck?

11. Was ist der Unterschied zwischen den Angaben {3} und {3,5} in einem regulären Ausdruck?
12. Wofür stehen die Zeichenklassen \d, \w und \s in einem regulären Ausdruck?
13. Wofür stehen die Zeichenklassen \D, \W und \S in einem regulären Ausdruck?
14. Wie sorgen Sie dafür, dass ein regulärer Ausdruck nicht zwischen Groß- und Kleinschreibung unterscheidet?
15. Wofür steht das Zeichen . normalerweise? Wofür steht es, wenn Sie re.DOTALL als zweites Argument an re.compile() übergeben?
16. Was ist der Unterschied zwischen .\* und .\*??
17. Welche Zeichenklasse geben Sie an, um nach einer Übereinstimmung mit allen Ziffern und Kleinbuchstaben zu suchen?
18. Gegeben sei numRegex = re.compile(r'\d+'). Was gibt numRegex.sub('X', '12 drummers, 11 pipers, five rings, 3 hens') zurück?
19. Was können Sie tun, wenn Sie re.VERBOSE als zweites Argument an re.compile() übergeben?
20. Wie schreiben Sie einen regulären Ausdruck, um nach Zahlen zu suchen, die wie im englischen Sprachraum üblich Kommas als Tausendertrennzeichen verwenden? Dieser reguläre Ausdruck muss folgende Zahlen finden können:

- '42'
- '1,234'
- '6,368,745'

Folgende Zahlen sollen aber nicht zu einer Übereinstimmung führen:

- '12,34,567' (nur zwei Ziffern zwischen den Kommas)
- '1234' (Tausenderzahl ohne Komma)

21. Wie schreiben Sie einen regulären Ausdruck, der die vollständigen Namen aller Personen mit dem Nachnamen Nakamoto findet? Sie können voraussetzen, dass der Vorname vor dem Nachnamen steht, immer ein einzelnes Wort ist und mit einem Großbuchstaben beginnt. Der Ausdruck muss also folgende Strings finden können:

- 'Satoshi Nakamoto'
- 'Alice Nakamoto'
- 'RoboCop Nakamoto'

Folgende Namen sollen aber nicht zu einer Übereinstimmung führen:

- 'satoshi Nakamoto' (Vorname mit kleinem Anfangsbuchstaben)
- 'Mr. Nakamoto' (das vorausgehende Wort enthält ein Zeichen, das kein Buchstabe ist)
- 'Nakamoto' (kein Vorname)
- 'Satoshi nakamoto' (Nachname mit kleinem Anfangsbuchstaben)

22. Wie schreiben Sie einen regulären Ausdruck, der Sätze mit dem folgenden Aufbau findet: Das erste Wort ist entweder *Alice*, *Bob* oder *Carol*, das zweite Wort ist entweder *eats*, *pets* oder *throws* und das dritte ist *apples*, *cats* oder *baseball*. Die Sätze müssen mit einem Punkt enden und es wird nicht nach Groß- und Kleinschreibung unterschieden. Der Ausdruck muss also folgende Sätze finden können:

- 'Alice eats apples.'
- 'Bob pets cats.'
- 'Carol throws baseballs.'
- 'Alice throws Apples.'
- 'BOB EATS CATS.'

Folgende Sätze sollen aber nicht zu einer Übereinstimmung führen:

- 'RoboCop eats apples.'
- 'ALICE THROWS FOOTBALLS.'
- 'Carol eats 7 cats.'

## Übungsprojekte

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erfüllen:

### Passwortstärke ermitteln

Schreiben Sie eine Funktion, die mithilfe regulärer Ausdrücke untersucht, ob der übergebene String ein starkes Passwort darstellt. Ein starkes Passwort muss mindestens acht Zeichen lang sein und sowohl Groß- und Kleinbuchstaben als auch mindestens eine Ziffer enthalten. Es kann sein, dass Sie den String anhand mehrerer regulärer Ausdrücke überprüfen müssen, um seine Stärke zu bestimmen.

### Regex-Version von strip()

Schreiben Sie eine Funktion, die einen String annimmt und das Gleiche macht wie die Stringmethode `strip()`. Wird außer dem String kein anderes Argument übergeben, sollen alle Weißraumzeichen vom Anfang und Ende des Strings entfernt werden. Andernfalls wird das im zweiten Argument übergebene Zeichen vom Anfang und Ende des Strings entfernt.

# 8

## Dateien lesen und schreiben

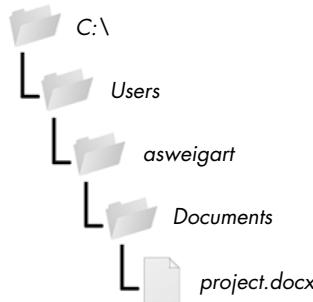


Variablen sind eine praktische Einrichtung, um Daten festzuhalten, während das Programm ausgeführt wird, aber wenn die Daten erhalten bleiben sollen, nachdem das Programm beendet ist, müssen Sie sie in einer Datei speichern. Sie können sich den Inhalt einer Datei als einen einzigen Stringwert vorstellen, dessen Größe durchaus in Gigabyte misst. In diesem Kapitel erfahren Sie, wie Sie in Python Dateien auf der Festplatte erstellen, lesen und speichern.

### Dateien und Dateipfade

Die beiden wichtigsten Merkmale einer Datei sind der *Dateiname* (der gewöhnlich als einzelnes Wort geschrieben wird) und der *Pfad*, der den Speicherort der Datei auf dem Computer angibt. Auf meinem Windows-7-Laptop habe ich beispielsweise eine Datei mit dem Namen *projects.docx* im Pfad C:\Users\asweigart\Documents. Der Teil des Dateinamens hinter dem Punkt wird als die *Dateinamenerweiterung* oder *Endung* bezeichnet und gibt den Typ der Datei an. Bei *project.docx* handelt es sich um ein Word-Dokument und *Users, asweigart* und

*Documents* sind *Ordner* (oder Verzeichnisse). Ordner können Dateien sowie andere Ordner enthalten. Beispielsweise befindet sich *projects.docx* im Ordner *Documents*, der wiederum im Ordner *asweigart* innerhalb des Ordners *Users* steckt. Diese Ordnerstruktur sehen Sie in Abb. 8–1.



**Abb. 8–1** Eine Datei in einer Ordnerhierarchie

Die Angabe C:\ im Pfad ist der *Wurzelordner* oder (unter Windows) *Stammordner*, der alle anderen Ordner enthält. Unter Windows heißt dieser Stammordner C:\ und wird auch als »Laufwerk C:« bezeichnet. Der Wurzelordner unter OS X und Linux ist /. In diesem Buch verwende ich den Windows-Stammordner C:\. Wenn Sie die Beispiele in einer interaktiven Shell unter OS X oder Linux eingeben, müssen Sie diese Angabe durch / ersetzen.

Zusätzliche Datenträger oder *Volumes*, etwa ein DVD-Laufwerk oder ein USB-Stick, werden in den verschiedenen Betriebssystemen unterschiedlich dargestellt. Unter Windows erscheinen sie als zusätzliche, mit Buchstaben gekennzeichnete Stammlaufwerke wie D:\ oder E:\, unter OS X als neue Ordner im Ordner /Volumes und unter Linux als neue Ordner unter /mnt (»mount«, was das Bereitstellen oder Einhängen eines solchen Laufwerks bezeichnet). Beachten Sie auch, dass bei Datei- und Ordnernamen unter Linux zwischen Groß- und Kleinschreibung unterschieden wird, unter Windows und OS X jedoch nicht.

### Backslash unter Windows und Schrägstrich unter OS X und Linux

Unter Windows werden Pfade mit Backslashes, also umgekehrtem Schrägstrich (\), als Trennzeichen zwischen den Ordnernamen geschrieben, unter OS X und Linux jedoch mit einem normalen Schrägstrich (/). Wenn Ihre Programme auf allen Betriebssystemen laufen sollen, müssen Sie Ihre Python-Skripts für beide Fälle auslegen.

Zum Glück lässt sich das mit der Funktion `os.path.join()` leicht erledigen. Wenn Sie ihr die Stringwerte der einzelnen Datei- und Ordnernamen eines Pfads übergeben,

gibt sie den Dateipfad als String mit den richtigen Trennzeichen für das vorliegende Betriebssystem zurück. Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> import os  
>>> os.path.join('usr', 'bin', 'spam')  
'usr\\bin\\spam'
```

In der interaktiven Shell auf Windows gibt `os.path.join('usr', 'bin', 'spam')` wie oben angegeben `'usr\\bin\\spam'` zurück (mit doppelten Backslashes, da jeder Backslash mit einem zweiten maskiert werden muss). Auf OS X und Linux dagegen ergibt sich der String `'usr/bin/spam'`.

Die Funktion `os.path.join()` ist sehr nützlich, wenn Sie Strings für Dateinamen erstellen müssen – was häufig vorkommt, da Sie den in diesem Kapitel vorgestellten Funktionen zur Dateibearbeitung solche Strings übergeben müssen. Der folgende Beispielcode hängt die einzelnen Einträge aus einer Liste von Dateinamen an einen Ordnerpfad an:

```
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']  
>>> for filename in myFiles:  
    print(os.path.join('C:\\\\Users\\\\asweigart', filename))  
  
C:\\Users\\asweigart\\accounts.txt  
C:\\Users\\asweigart\\details.csv  
C:\\Users\\asweigart\\invite.docx
```

## Das aktuelle Arbeitsverzeichnis

Jedes Programm, das auf einem Computer läuft, hat ein *aktuelles Arbeitsverzeichnis*. Bei allen Dateinamen und Pfaden, die nicht mit dem Stammordner beginnen, wird angenommen, dass sie sich in diesem Arbeitsverzeichnis befinden. Um den Stringwert des aktuellen Arbeitsverzeichnisses abzurufen, verwenden Sie die Funktion `os.getcwd()` (wobei `cwd` für *current working directory* steht), und um es zu ändern die Funktion `os.chdir()`:

```
>>> import os  
>>> os.getcwd()  
'C:\\Python34'  
>>> os.chdir('C:\\Windows\\System32')  
>>> os.getcwd()  
'C:\\Windows\\System32'
```

Hier ist das aktuelle Arbeitsverzeichnis zunächst `C:\\Python34`, weshalb der Dateiname `project.docx` auf `C:\\Python34\\project.docx` verweist. Wenn wir das aktuelle

Arbeitsverzeichnis anschließend in C:\Windows ändern, wird *project.docx* als C:\Windows\project.docx interpretiert.

Wenn Sie versuchen, zu einem Verzeichnis zu wechseln, das gar nicht existiert, gibt Python eine Fehlermeldung aus:

```
>>> os.chdir('C:\\\\ThisFolderDoesNotExist')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    os.chdir('C:\\\\ThisFolderDoesNotExist')
FileNotFoundError: [WinError 2] The system cannot find the file specified:
'C:\\\\ThisFolderDoesNotExist'
```

### Hinweis

*Ordner* ist zwar die moderne Bezeichnung für ein Verzeichnis, doch wird das aktuelle Arbeitsverzeichnis (oder einfach das Arbeitsverzeichnis) normalerweise genauso bezeichnet und nicht als aktueller Arbeitsordner.

### Absolute und relative Pfade

Es gibt zwei Möglichkeiten, um einen Dateipfad anzugeben:

- Als *absoluter Pfad*, der stets mit dem Stammordner beginnt.
- Als *relativer Pfad*, der relativ zum aktuellen Arbeitsverzeichnis des Programms angegeben ist.

In Pfaden finden Sie oft auch die Angaben . und ... Dabei handelt es sich nicht um echte Ordner, sondern um besondere Bezeichnungen. Der einzelne Punkt . steht dabei für das vorliegende Verzeichnis, zwei Punkte .. für den Elternordner.

Abb. 8–2 zeigt Beispiele für Ordner und Dateien. Wenn C:\bacon das aktuelle Arbeitsverzeichnis ist, dann lauten die relativen Pfade für die dargestellten Ordner und Dateien wie in dem Bild angegeben.

	Relative Pfade	Absolute Pfade
C:\	..\ .	C:\
Aktuelles Arbeitsverzeichnis → bacon	.\ .\ spam.txt	C:\bacon
fizz	.fizz\ spam.txt	C:\bacon\fizz
spam.txt	.spam.txt	C:\bacon\fizz\spam.txt
eggs	..eggs spam.txt	C:\bacon\spam.txt
spam.txt	..eggs\spam.txt spam.txt	C:\eggs\spam.txt
	..spam.txt	C:\spam.txt

Abb. 8–2 Relative und absolute Pfade für die Ordner und Dateien im Arbeitsverzeichnis C:\bacon

Die Angabe .\ zu Beginn eines relativen Pfads ist optional. Beispielsweise verweisen sowohl .\spam.txt als auch spam.txt auf dieselbe Datei.

### Neue Ordner mit os.makedirs() erstellen

Mithilfe der Funktion `os.makedirs()` können Ihre Programme neue Ordner erstellen (wobei sich *dirs* in dem Funktionsnamen auf die alternative Bezeichnung *directories* für Verzeichnisse bezieht):

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

Dadurch wird nicht nur der Ordner *C:\delicious* erstellt, sondern auch der Ordner *walnut* innerhalb von *C:\delicious* und der Ordner *waffles* innerhalb von *C:\delicious\walnut*. Die Funktion `os.makedirs()` erstellt also alle erforderlichen Zwischenordner, um dafür zu sorgen, dass der vollständige Pfad existiert. Diese Ordnerhierarchie sehen Sie in Abb. 8–3.

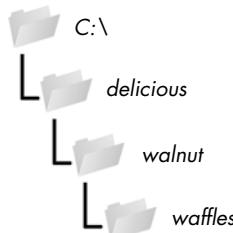


Abb. 8–3 Das Ergebnis von `os.makedirs('C:\\delicious\\walnut\\waffles')`

## Das Modul os.path

Das Modul `os.path` enthält viele nützliche Funktionen für den Umgang mit Datei- und Pfadnamen. Bereits kennengelernt haben Sie die Funktion `os.path.join()`, mit der Sie Pfade passend zum vorliegenden Betriebssystem zusammenstellen können. Da es sich bei `os.path` um ein Modul innerhalb des Moduls `os` handelt, können Sie es einfach durch `import os` importieren. Wenn Sie in Ihrem Programm mit Dateien, Ordnern oder Pfaden arbeiten müssen, schlagen Sie die Beispiele in diesem Abschnitt nach. Die komplette Dokumentation des Moduls `os.path` finden Sie auf der Python-Website unter <http://docs.python.org/3/library/os.path.html>.

### Hinweis

Für die meisten Beispiele in diesem Abschnitt ist das Modul `os` erforderlich, weshalb Sie es zu Beginn jedes Skripts und bei jedem Neustart von IDLE importieren müssen. Andernfalls erhalten Sie die Fehlermeldung `NameError: name 'os' is not defined`.

### Absolute und relative Pfade verwenden

Das Modul `os.path` enthält Funktionen, die den absoluten Pfad zu einem relativen Pfad zurückgeben oder prüfen, ob ein gegebener Pfad ein absoluter Pfad ist.

- Die Funktion `os.path.abspath(path)` gibt den absoluten Pfad des Arguments als String zurück. Das bietet eine einfache Möglichkeit, um einen relativen in einen absoluten Pfad umzuwandeln.
- Die Funktion `os.path.isabs(path)` gibt `True` zurück, wenn das Argument ein absoluter Pfad ist, und `False`, wenn es sich um einen relativen Pfad handelt.
- Die Funktion `os.path.relpath(path, start)` gibt den String des relativen Pfads vom Ausgangspunkt (`start`) bis zu `path` zurück. Wenn Sie `start` nicht angeben, wird das aktuelle Arbeitsverzeichnis als Ausgangspunkt genommen.

Probieren Sie diese Funktionen in der interaktiven Shell aus:

```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath('.\\\\Scripts')
'C:\\Python34\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

Da C:\Python34 beim Aufruf von `os.path.abspath()` das Arbeitsverzeichnis war, steht der Punkt für den absoluten Pfad C:\Python34.

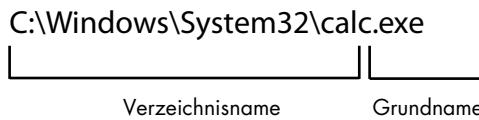
## Hinweis

Die Ordner- und Dateistruktur auf Ihrem System unterscheidet sich natürlich von der auf meinem Computer. Daher werden Sie manche Beispiele in diesem Kapitel nicht exakt nachvollziehen können. Versuchen Sie aber, die Beispiele anhand der Ordner auf Ihrem Computer zu verfolgen.

Probieren Sie auch folgende Aufrufe von `os.path.relpath()` in der interaktiven Shell aus:

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
>>> os.getcwd()
'C:\\Python34'
```

Die Funktion `os.path.dirname(path)` gibt einen String mit allem zurück, was vor dem letzten (umgekehrten) Schrägstrich im Argument `path` steht, die Funktion `os.path.basename(path)` dagegen einen String mit allem, was hinter diesem letzten Schrägstrich steht. Verzeichnisname (*dir name*) und Grundname (*base name*) eines Pfads sind in Abb. 8–4 dargestellt.



**Abb. 8-4** Der Grundname folgt auf den letzten Schrägstrich im Pfad und ist mit dem Dateinamen identisch. Alles, was links von dem letzten Schrägstrich steht, gehört dagegen zum Verzeichnisnamen.

Geben Sie beispielsweise Folgendes in die interaktive Shell ein:

```
>>> path = 'C:\\Windows\\\\System32\\\\calc.exe'  
>>> os.path.basename(path)  
'calc.exe'  
>>> os.path.dirname(path)  
'C:\\Windows\\\\System32'
```

Wenn Sie sowohl den Verzeichnis- als auch den Grundnamen eines Pfads benötigen, können Sie `os.path.split()` aufrufen, um einen Tupelwert mit diesen beiden Strings zu erhalten:

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'  
>>> os.path.split(calcFilePath)  
('C:\\Windows\\System32', 'calc.exe')
```

Das gleiche Tupel können Sie auch erstellen, indem Sie `os.path.dirname()` und `os.path.basename()` aufrufen und die Rückgabewerte in ein Tupel aufnehmen:

```
>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))  
('C:\\Windows\\System32', 'calc.exe')
```

Allerdings ist `os.path.split()` eine praktische Abkürzung.

Beachten Sie aber, dass `os.path.split()` keinen Dateipfad entgegennimmt und daraus eine Liste der Strings für die einzelnen Ordner zurückgibt. Für diesen Zweck müssen Sie den Pfad mit der Stringmethode `split()` zerlegen und ihr die Variable `os.path.sep` übergeben, in der das Ordner trennzeichen für das vorliegende Betriebssystem gespeichert ist.

Betrachten Sie dazu das folgende Beispiel in der interaktiven Shell:

```
>>> calcFilePath.split(os.path.sep)  
['C:', 'Windows', 'System32', 'calc.exe']
```

Unter OS X und Linux steht am Anfang der zurückgegebenen Liste ein leerer String:

```
>>> '/usr/bin'.split(os.path.sep)  
['', 'usr', 'bin']
```

Die Methode `split()` gibt eine Liste der einzelnen Teile des Pfads zurück. Damit sie auf allen Betriebssystemen korrekt funktioniert, müssen Sie ihr `os.path.sep` übergeben.

## Dateigrößen und Ordnerinhalte ermitteln

Nachdem Sie nun wissen, wie Sie mit Dateipfaden umgehen müssen, können Sie Informationen über einzelne Dateien und Ordner abrufen. Das Modul `os` enthält Funktionen, mit denen Sie die Größe eines Ordners in Byte ermitteln und feststellen können, welche Dateien und Ordner sich innerhalb eines gegebenen Ordners befinden.

- Die Funktion `os.path.getsize(path)` gibt die Größe der im Argument `path` übergebenen Datei in Byte zurück.
- Die Funktion `os.listdir(path)` gibt eine Liste der Namenstrings aller Dateien und Ordner zurück, die sich in dem als `path` übergebenen Pfad befinden. (Beachten Sie, dass sich diese Funktion nicht im Modul `os.path`, sondern in `os` befindet.)

Wenn ich diese Funktionen in der interaktiven Shell ausprobiere, erhalte ich folgende Ergebnisse:

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
776192
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--schnipp--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

Wie Sie sehen, ist das Programm *calc.exe* auf meinem Computer 776.192 Bytes groß und im Ordner C:\Windows\System32 befinden sich eine ganze Menge Dateien. Wenn ich die Gesamtgröße aller Dateien in diesem Verzeichnis herausfinden möchte, kann ich `os.path.getsize()` und `os.listdir()` zusammen einsetzen:

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
    totalSize = totalSize +
    os.path.getsize(os.path.join('C:\\Windows\\System32', filename))

>>> print(totalSize)
1117846456
```

Dieser Code durchläuft alle Dateien im Ordner C:\Windows\System32 und erhöht dabei die Variable `totalSize` um die Größe der jeweiligen Datei. Beachten Sie, dass ich hier beim Aufruf von `os.path.getsize()` die Funktion `os.path.join()` verwende, um den Ordnernamen mit dem Namen der aktuellen Datei zu verknüpfen. Der von `os.path.getsize()` zurückgegebene Integer wird dann zu dem Wert in `totalSize` addiert. Nachdem alle Dateien durchlaufen wurden, gibt das Programm `totalSize` aus, um die Gesamtgröße aller Dateien im Ordner C:\Windows\System32 anzuzeigen.

## Die Gültigkeit von Pfaden prüfen

Viele Python-Funktionen werden mit einer Fehlermeldung beendet, wenn Sie ihnen einen Pfad übergeben, den es gar nicht gibt. Um das zu verhindern, finden Sie im Modul `os.path` Funktionen, mit denen Sie prüfen können, ob ein gegebener Pfad existiert und ob es sich dabei um eine Datei oder einen Ordner handelt.

- Die Funktion `os.path.exists(path)` gibt `True` zurück, wenn die Datei oder der Ordner im Argument vorhanden ist, und anderenfalls `False`.
- Die Funktion `os.path.isfile(path)` gibt `True` zurück, wenn das Pfadargument vorhanden und eine Datei ist, und anderenfalls `False`.
- Die Funktion `os.path.isdir (path)` gibt `True` zurück, wenn das Pfadargument vorhanden und ein Ordner ist, und anderenfalls `False`.

Beim Ausprobieren in der interaktiven Shell habe ich folgende Ergebnisse erhalten:

```
>>> os.path.exists('C:\\Windows')
True
>>> os.path.exists('C:\\some_made_up_folder')
False
>>> os.path.isdir('C:\\Windows\\System32')
True
>>> os.path.isfile('C:\\Windows\\System32')
False
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')
False
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')
True
```

Mit der Funktion `os.path.exists()` können Sie auch prüfen, ob an den Computer ein DVD- oder Flash-Laufwerk angeschlossen ist. Beispielsweise kann ich auf meinem Windows-Rechner wie folgt herausfinden, ob ein Flash-Laufwerk mit dem Volume-Namen *D:\* vorhanden ist:

```
>>> os.path.exists('D:\\')
False
```

Hoppla! Da habe ich wohl vergessen, mein Flash-Laufwerk anzuschließen.

## Dateien lesen und schreiben

Wenn Sie im Umgang mit Ordnern und relativen Pfaden sicher sind, können Sie die Speicherorte der Daten angeben, in denen Sie lesen und schreiben wollen. Die in diesem Abschnitt beschriebenen Funktionen dienen zur Arbeit mit *reinen Textdateien*. Sie enthalten nur einfache Textzeichen ohne Informationen über Schriftart, Schriftgröße oder Farbe. Beispiele dafür sind Textdateien mit der Endung *.txt* und Python-Skriptdateien mit der Erweiterung *.py*. Geöffnet werden können sie mit Programmen wie dem Windows-Editor oder der OS-X-Anwendung *TextEdit*. In Python können Sie Programme schreiben, die die Inhalte solcher reinen Textdateien lesen und als ganz normale Stringwerte behandeln können.

*Binärdateien* dagegen sind alle anderen Dateitypen, z. B. die Dokumente von Textverarbeitungsprogrammen, PDF-Dokumente, Bilder, Arbeitsblätter und ausführbare Programme. Wenn Sie eine Binärdatei im Editor oder in *TextEdit* öffnen, sehen Sie nur einen Haufen wirrer, sinnloser Zeichen wie in Abb. 8–5.

Da die verschiedenen Arten von Binärdateien jeweils auf ihre eigene Art und Weise behandelt werden müssen, gehe ich in diesem Buch nicht darauf ein, wie rohe Binärdateien gelesen und geschrieben werden. Zum Glück gibt es eine Reihe von



**Abb. 8–5** Das Windows-Programm calc.exe im Editor

Modulen, die die Arbeit mit Binärdateien erleichtern. Eines davon, das Modul `shelve`, lernen Sie in diesem Kapitel noch kennen.

Um Dateien in Python zu lesen oder zu schreiben, sind drei Schritte erforderlich:

1. Rufen Sie die Funktion `open()` auf, um ein `File`-Objekt zurückzugeben.
2. Rufen Sie die Funktion `read()` oder `write()` für das `File`-Objekt auf.
3. Schließen Sie die Datei, indem Sie die Methode `close()` für das `File`-Objekt aufrufen.

### Dateien mit der Funktion `open()` öffnen

Um mit der Funktion `open()` eine Datei zu öffnen, müssen Sie ihr den String mit dem Pfad der gewünschten Datei übergeben. Dabei kann es sich sowohl um einen absoluten als auch um einen relativen Pfad handeln. Die Funktion `open()` gibt ein `File`-Objekt zurück.

Probieren Sie das aus, indem Sie mit dem Editor oder mit `TextEdit` eine Textdatei namens `hello.txt` erstellen. Geben Sie **Hello world!** als Inhalt dieser Datei ein und speichern Sie sie in Ihrem Benutzerordner. Unter Windows geben Sie dann Folgendes in die interaktive Shell ein:

```
>>> helloFile = open('C:\\\\Users\\\\Benutzerordner\\\\hello.txt')
```

Unter OS X verwenden Sie stattdessen folgenden Befehl:

```
>>> helloFile = open('/Users/Benutzerordner/hello.txt')
```

ersetzen Sie dabei `Benutzerordner` durch Ihren Benutzernamen auf dem Computer. Beispielsweise verwende ich auf meinem Windows-Rechner, wo mein Benutzername `asweigart` lautet, die Eingabe `'C:\\\\Users\\\\asweigart\\\\hello.txt'`.

Beide Befehle öffnen die Datei im Reintext-Lesemodus oder kurz *Lesemodus*. In diesem Modus erlaubt Python Ihnen nur, Daten in der Datei zu lesen; Daten schreiben oder auf irgendeine Weise ändern können Sie dagegen nicht. Dies ist der Standardmodus für Dateien, die Sie in Python öffnen. Wenn Sie sich nicht auf diese Standardeinstellung verlassen wollen, können Sie den Modus auch ausdrücklich angeben, indem Sie als zweites Argument von `open()` den Stringwert '`r`' übergeben. Die Funktionsaufrufe `open('/Users/asweigart/hello.txt', 'r')` und `open('/Users/asweigart/hello.txt')` bewirken daher das Gleiche.

Der Aufruf von `open()` gibt ein `File`-Objekt zurück. Ein solches Objekt steht für eine Datei auf Ihrem Computer. Es handelt sich dabei einfach nur um eine weitere Art von Werten in Python, ebenso wie Listen oder Dictionaries. Im vorstehenden Beispiel wird das `File`-Objekt in der Variablen `helloFile` gespeichert. Wenn Sie nun in der Datei lesen oder schreiben wollen, können Sie das tun, indem Sie entsprechende Methoden für das `File`-Objekt in `helloFile` aufrufen.

### Die Inhalte einer Datei lesen

Da Sie nun über ein `File`-Objekt verfügen, können Sie darin lesen. Wenn Sie den gesamten Inhalt der Datei als Stringwert lesen möchten, verwenden Sie die Methode `read()` des `File`-Objekts. Bleiben wir bei unserem Beispiel, bei dem das `File`-Objekt für die Datei `hello.txt` in `helloFile` gespeichert ist. Geben Sie nun Folgendes in die interaktive Shell ein:

```
>>> helloContent = helloFile.read()  
>>> helloContent  
'Hello world!'
```

Stellen Sie sich den Inhalt einer Datei als einen einzigen, umfangreichen Stringwert vor. Die Methode `read()` gibt diesen String zurück.

Alternativ können Sie mit der Methode `readlines()` auch eine Liste von Stringwerten aus der Datei abrufen, wobei jeder String für eine Textzeile steht. Um das auszuprobieren, legen Sie die Datei `sonnet29.txt` in demselben Verzeichnis an wie `hello.txt` und schreiben Folgendes hinein:

```
When, in disgrace with fortune and men's eyes,  
I all alone beweep my outcast state,  
And trouble deaf heaven with my bootless cries,  
And look upon myself and curse my fate,
```

Achten Sie darauf, die vier Zeilen durch Zeilenumbrüche zu trennen. Geben Sie in der interaktiven Shell dann Folgendes ein:

```
>>> sonnetFile = open('sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone beweep my
outcast state,\n', And trouble deaf heaven with my bootless cries,\n', And
look upon myself and curse my fate,']
```

Alle einzelnen Stringwerte enden mit dem Zeilenumbruchzeichen \n, ausgenommen die letzte Zeile der Datei. Häufig ist es einfacher, mit einer Liste von Strings zu arbeiten als mit einem einzigen, riesigen String.

## Dateien schreiben

In Python können Sie auch Inhalte in eine Datei schreiben, ähnlich wie Sie mit print() Strings auf den Bildschirm »schreiben«. Allerdings ist es nicht möglich, in eine Datei zu schreiben, die im Lesemodus geöffnet ist. Stattdessen müssen Sie sie im *Schreibmodus* oder im *Anhängemodus* öffnen.

Im Schreibmodus wird die vorhandene Datei überschrieben und ganz neu mit Text gefüllt, ähnlich wie Sie einen Variablenwert mit einem neuen Wert überschreiben. Um eine Datei in diesem Modus zu öffnen, übergeben Sie 'w' als zweites Argument der Funktion open(). Im Anhängemodus dagegen wird neuer Text am Ende der Datei angehängt, so wie Sie neue Elemente hinten zu einer Liste hinzufügen, anstatt sie komplett zu überschreiben. Hierzu verwenden Sie 'a' als zweites Argument von open().

Wenn es keine Datei mit dem an open() übergebenen Namen gibt, wird sowohl im Schreib- als auch im Anhängemodus eine neue, leere Datei erstellt. Nach dem Lesen oder Schreiben in einer Datei müssen Sie die Methode close() aufrufen, bevor Sie die Datei wieder öffnen können.

Sehen wir uns all diese verschiedenen Möglichkeiten nun im Zusammenhang an. Dazu probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello world!
Bacon is not a vegetable.
```

Als Erstes öffnen wir hier *bacon.txt* im Schreibmodus. Da es eine Datei dieses Namens noch nicht gibt, wird sie von Python erstellt. Danach rufen wir die Funktion `write()` für die geöffnete Datei auf und übergeben ihr das Stringargument `'Hello world! \n'`, wodurch dieser String in die Datei geschrieben wird. Außerdem wird die Anzahl der geschriebenen Zeichen einschließlich des Zeilenumbruchs ausgegeben. Daraufhin schließen wir die Datei.

Um Text zu den bestehenden Inhalten der Datei hinzuzufügen, ohne den eben geschriebenen String zu ersetzen, öffnen wir die Datei anschließend im Anhänge-Modus, schreiben `'Bacon is not a vegetable.'` in die Datei und schließen sie. Als Letztes geben wir den Inhalt der Datei auf dem Bildschirm aus. Dazu öffnen wir sie im Standardmodus, dem Lesemodus, rufen `read()` auf, speichern das resultierende File-Objekt in `content`, schließen die Datei und geben `content` aus.

Beachten Sie, dass die Methode `write()` im Gegensatz zu `print()` am Ende eines Strings nicht automatisch einen Zeilenumbruch einfügt. Das müssen Sie manuell tun.

## Variablen mit dem Modul `shelve` speichern

Mit dem Modul `shelve` können Sie in Ihren Python-Programmen Variablen in binären »Shelf-Dateien« speichern. Dadurch kann das Programm die Daten der Variablen von der Festplatte wiederherstellen. Dieses Modul ermöglicht es Ihnen, Speichern- und Öffnen-Funktionen zu Ihren Programmen hinzuzufügen. Wenn ein Benutzer in einem Programm beispielsweise Konfigurationseinstellungen vorgenommen hat, kann er sie in einer Shelf-Datei speichern, sodass das Programm sie bei der nächsten Ausführung lädt.

Geben Sie Folgendes in die interaktive Shell ein:

```
>>> import shelve  
>>> shelfFile = shelve.open('mydata')  
>>> cats = ['Zophie', 'Pooka', 'Simon']  
>>> shelfFile['cats'] = cats  
>>> shelfFile.close()
```

Um Daten mithilfe des Moduls `shelve` zu lesen und zu schreiben, müssen Sie es zunächst importieren. Rufen Sie dann die Funktion `shelve.open()` auf, übergeben Sie ihr einen Dateinamen und speichern Sie den zurückgegebenen Shelf-Wert in einer Variablen. Nun können Sie an diesem Shelf-Wert Änderungen vornehmen wie an einem Dictionary. Abschließend rufen Sie `close()` für den Shelf-Wert auf. In unserem Beispiel wird der Shelf-Wert in `shelfFile` gespeichert. Wir erstellen die Liste `cats` und speichern diese Liste mit `shelfFile['cats']` als Wert zu dem Schlüs-

sel 'cats' in `shelfFile` (wie in einem Dictionary). Danach rufen wir `close()` für `shelfFile` auf.

Wenn Sie diesen Code unter Windows ausführen, stehen anschließend drei neue Dateien in Ihrem Arbeitsverzeichnis: *mydata.bak*, *mydata.dat* und *mydata.dir*. Unter OS X dagegen wird nur eine einzige Datei namens *mydata.db* erstellt.

Diese Binärdateien enthalten die Daten, die Sie in dem Shelf gespeichert haben. Das genaue Format dieser Dateien brauchen Sie nicht zu kennen. Sie müssen nur wissen, was das Modul `shelve` tut, aber nicht, wie es das im Einzelnen macht. Dieses Modul nimmt Ihnen die Verantwortung dafür ab, die Programmdaten in einer Datei zu speichern.

Mit dem Modul `shelve` können Sie die Shelf-Dateien in Ihrem Programm auch wieder öffnen und die Daten daraus lesen. Dabei ist es nicht erforderlich, ausdrücklich den Lese- oder Schreibmodus zu verlangen, denn nach dem Öffnen ist beides möglich. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

Hier öffnen wir die Shelf-Dateien, um zu prüfen, ob sie nach wie vor die richtigen Daten enthalten. `shelfFile['cats']` gibt genau die Liste zurück, die wir zuvor gespeichert haben. Da wir nun wissen, dass die richtigen Daten gespeichert sind, rufen wir `close()` auf.

Wie für Dictionarys gibt es auch für Shelf-Werte die Methoden `keys()` und `values()`, die listenähnliche Werte der Schlüssel bzw. Werte in dem Shelf zurückgeben. Um daraus echte Listen zu machen, müssen Sie sie an die Funktion `list()` übergeben, wie das folgende Beispiel zeigt:

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()
```

Reiner Text ist sinnvoll für Dateien, die in einem Texteditor wie dem Windows-Editor oderTextEdit gelesen werden sollen. Wenn Sie dagegen Daten aus Ihrem Python-Programm speichern wollen, sollten Sie das Modul `shelve` verwenden.

## Variablen mit der Funktion pprint.pformat() speichern

Im Abschnitt »Saubere Ausgabe« in Kapitel 5 haben Sie zwei Funktionen des Moduls `pprint` kennengelernt. Während `pprint.pprint()` den Inhalt einer Liste oder eines Dictionarys in übersichtlicher Form auf dem Bildschirm anzeigt, gibt `pprint.pformat()` den Text als String aus, der nicht nur übersichtlich formatiert ist, sondern auch syntaktisch korrekter Python-Code ist. Wenn Sie eine Variable, die ein Dictionary enthält, für den späteren Gebrauch speichern wollen, können Sie mit `pprint.pformat()` einen String ausgeben und diesen dann in eine `.py`-Datei schreiben. Diese Datei ist dann ein eigenes Modul, das Sie später jederzeit importieren können, um die darin gespeicherte Variable zu verwenden.

Um das zu veranschaulichen, geben Sie Folgendes in die interaktive Shell ein:

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc':
   'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

Als Erstes importieren wir hier `pprint`, damit wir `pprint.pformat()` verwenden können. In der Variablen `cats` ist eine Liste gespeichert, deren einzelne Elemente wiederum Dictionarys sind. Wenn wir die Shell schließen, würde diese Liste normalerweise verloren gehen. Um das zu verhindern, geben wir sie mithilfe von `pprint.pformat()` als String aus, den wir dann ganz einfach als Datei namens `myCats.py` speichern können.

Die Module, die wir mit der Anweisung `import` importieren, sind nichts anderes als Python-Skripte. Wenn wir den von `pprint.pformat()` zurückgegebenen String in einer `.py`-Datei speichern, ist diese Datei ebenfalls ein Modul, das wir wie alle anderen importieren können.

Python-Skripte wiederum sind nichts anderes als Textdateien mit der Endung `.py`. Daher können Python-Programme andere Python-Programme generieren, die Sie dann in Ihre Skripts importieren können.

```
>>> import myCats
>>> myCats.cats
[{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> myCats.cats[0]
{'name': 'Zophie', 'desc': 'chubby'}
>>> myCats.cats[0]['name']
'Zophie'
```

Gegenüber der Speicherung von Variablen mit dem Modul `shelve` bietet die Speicherung als `.py`-Datei den Vorteil, dass sich der Inhalt dieser Datei mit einem einfachen Texteditor lesen und bearbeiten lässt, da es sich schließlich um eine Textdatei handelt. In den meisten Anwendungen ist die Verwendung von `shelve` trotzdem die bevorzugte Möglichkeit, um Variablen zu speichern, denn nur einfache Datentypen wie Integer, Fließkommazahlen, Strings, Listen und Dictionarys können in eine einfache Textdatei geschrieben werden. Bei anderen Datentypen, beispielsweise bei File-Objekten, ist das dagegen nicht möglich.

## Projekt: Zufallsgenerator für Tests

Stellen Sie sich vor, Sie unterrichten Erdkunde in einer Klasse mit 35 Schülern und wollen einen Test über die Hauptstädte der US-Bundesstaaten schreiben. Allerdings sind in dieser Klasse ein paar Tunichtgute, sodass Sie befürchten müssen, dass einige der Schüler schummeln. Daher ordnen Sie die Fragen in jedem Test zufällig an, sodass kein Test dem anderen gleicht und daher niemand von seinem Nachbarn abschreiben kann. Es wäre allerdings ziemlich langwierig und langweilig, das manuell zu tun. Zum Glück kennen Sie sich aber schon ein bisschen mit Python aus.

Das Programm soll folgende Aufgaben erfüllen:

- 35 verschiedene Tests zusammenstellen
- Für jeden Test 50 Multiple-Choice-Fragen in zufälliger Reihenfolge erstellen
- Zu jeder Frage die korrekte Antwort und drei zufällige falsche Antworten in zufälliger Reihenfolge bereitstellen
- Die Testfragebogen in 35 Textdateien schreiben
- Die Lösungsbogen in 35 Textdateien schreiben

Der Code muss also Folgendes tun:

- Die Namen der Bundesstaaten und der zugehörigen Hauptstädte in einem Dictionary speichern
- Die Funktionen `open()`, `write()` und `close()` für die Textdateien mit den Frage- und den Lösungsbogen aufrufen
- Die Fragen und die möglichen Antworten mit `random.shuffle()` in eine zufällige Reihenfolge bringen

### Schritt 1: Die Daten für den Test in einem Dictionary speichern

Der erste Schritt besteht darin, ein Skelett für das Skript zu erstellen und darin die Daten aufzunehmen, die in dem Test abgefragt werden sollen. Erstellen Sie die Datei `randomQuizGenerator.py` und geben Sie darin folgenden Code ein:

```
#! python3
# randomQuizGenerator.py - Erstellt Testfragebogen mit Fragen und Antworten
# in zufälliger Reihenfolge sowie die zugehörigen Lösungsbogen

import random ❶

# Die abzufragenden Daten. Die Schlüssel sind die Bundesstaaten, die Werte
# deren Hauptstädte.
capitals = {'Alabama': 'Montgomery', 'Alaska': 'Juneau', 'Arizona': ❷
'Phoenix', 'Arkansas': 'Little Rock', 'California': 'Sacramento', 'Colorado':
'Denver', 'Connecticut': 'Hartford', 'Delaware': 'Dover', 'Florida': 'Talla-
hassee', 'Georgia': 'Atlanta', 'Hawaii': 'Honolulu', 'Idaho': 'Boise',
'Illinois': 'Springfield', 'Indiana': 'Indianapolis', 'Iowa': 'Des Moines',
'Kansas': 'Topeka', 'Kentucky': 'Frankfort', 'Louisiana': 'Baton Rouge',
'Maine': 'Augusta', 'Maryland': 'Annapolis', 'Massachusetts': 'Boston',
'Michigan': 'Lansing', 'Minnesota': 'Saint Paul', 'Mississippi': 'Jackson',
'Missouri': 'Jefferson City', 'Montana': 'Helena', 'Nebraska': 'Lincoln',
'Nevada': 'Carson City', 'New Hampshire': 'Concord', 'New Jersey': 'Trenton',
'New Mexico': 'Santa Fe', 'New York': 'Albany', 'North Carolina': 'Raleigh',
'North Dakota': 'Bismarck', 'Ohio': 'Columbus', 'Oklahoma': 'Oklahoma City',
'Oregon': 'Salem', 'Pennsylvania': 'Harrisburg', 'Rhode Island': 'Provi-
dence', 'South Carolina': 'Columbia', 'South Dakota': 'Pierre', 'Tennessee':
'Nashville', 'Texas': 'Austin', 'Utah': 'Salt Lake City', 'Vermont': 'Montpe-
lier', 'Virginia': 'Richmond', 'Washington': 'Olympia', 'West Virginia':
'Charleston', 'Wisconsin': 'Madison', 'Wyoming': 'Cheyenne'}
```

# Erstellt 35 Testfragebogen  
for quizNum in range(35): ❸  
 # TODO: Dateien für Frage- und Lösungsbogen erstellen

# TODO: Kopf für den Test schreiben

# TODO: Die Reihenfolge der Bundesstaaten durcheinanderwürfeln

# TODO: Alle 50 Staaten durchlaufen und für jeden eine Frage erstellen

Da dieses Programm die Fragen und Antworten willkürlich anordnen soll, müssen Sie das Modul `random` importieren (❶), um dessen Funktionen nutzen zu können. Die Variable `capitals` (❷) enthält ein Dictionary mit den Namen der US-Bundesstaaten als Schlüssel und deren Hauptstädten als Werte. Da Sie 35 Tests erstellen wollen, muss der Code, der die Dateien für die Frage- und Lösungsbogen generieren soll (zurzeit nur durch die `TODO`-Kommentare angezeigt), in einer `for`-Schleife mit 35 Iterationen stehen (❸). Wenn Sie eine andere Anzahl individueller Tests benötigen, können Sie die Zahl der Iterationen einfach ändern.

## Schritt 2: Die Fragebogendatei erstellen und die Fragen mischen

Beginnen wir nun damit, die TODO-Platzhalter zu ersetzen.

Der Code in der Schleife wird 35 Mal ausgeführt – einmal für jeden individuellen Test –, sodass Sie sich in der Schleife immer nur um einen Test auf einmal kümmern müssen. Als Erstes müssen Sie die Datei für den Test erstellen. Sie braucht einen eindeutigen Dateinamen und einen Standardkopf, in dem die Schüler später Name, Datum und Schuljahr eintragen. Anschließend brauchen Sie eine Liste der Bundesstaaten in zufälliger Reihenfolge, aus der Sie dann später die Fragen und Antworten für den Test entnehmen.

Ergänzen Sie *randomQuizGenerator.py* auf folgende Weise:

```
#! python3
# randomQuizGenerator.py - Erstellt Testfragebogen mit Fragen und Antworten
# in zufälliger Reihenfolge sowie die zugehörigen Lösungsbogen

-- schnipp --

# Erstellt 35 Testfragebogen
for quizNum in range(35):
    # Erstellt die Dateien für die Frage- und Lösungsbogen
    quizFile = open('capitalsquiz%s.txt' % (quizNum + 1), 'w')      ❶
    answerKeyFile = open('capitalsquiz_answers%s.txt' % (quizNum + 1), 'w')
    ❷

    # Schreibt den Kopf für den Test
    quizFile.write('Name:\n\nDate:\n\nPeriod:\n\n')      ❸
    quizFile.write(( ' ' * 20) + 'State Capitals Quiz (Form %s)' %
                   (quizNum + 1))
    quizFile.write('\n\n')

    # Würfelt die Reihenfolge der Bundesstaaten durcheinander
    states = list(capitals.keys())
    random.shuffle(states) ❹

    # TODO: Alle 50 Staaten durchlaufen und für jeden eine Frage erstellen
```

Die Dateinamen für die Tests folgen dem Muster *capitalsquiz<N>.txt*, wobei <N> die laufende Nummer des Tests ist. Abgeleitet wird diese Nummer von *quizNum*, dem Zähler der for-Schleife. Der Lösungsbogen für die Datei *capitalsquiz<N>.txt* befindet sich in der Textdatei *capitalsquiz\_answers<N>.txt*. Bei jedem Schleifendurchlauf wird der Platzhalter *%s* in 'capitalsquiz%s.txt' und 'capitalsquiz\_answers%s.txt' durch *(quizNum + 1)* ersetzt, sodass das erste Paar von Frage- und Lösungsbogen die Namen *capitalsquiz1.txt* und *capitalsquiz\_answers1.txt* trägt. Erstellt werden diese Dateien mit Aufrufen von *open()* in (❶) und (❷), wobei als zweites Argument '*w*' übergeben wird, um sie im Schreibmodus zu öffnen.

Die Anweisung `write()` bei (❸) erstellt den Kopf, den die Schüler später ausfüllen müssen. Schließlich wird mithilfe der Funktion `random.shuffle()` eine willkürlich durcheinandergewürfelte Liste der US-Bundesstaaten erstellt (❹). Diese Funktion ordnet die Werte in der ihr übergebenen Liste in zufälliger Reihenfolge an.

### Schritt 3: Die Auswahl der möglichen Antworten zusammenstellen

Als Nächstes müssen Sie die angebotenen Antworten A bis D für die einzelnen Fragen zusammenstellen. Sie benötigen eine weitere `for`-Schleife, um den Inhalt jeder der 50 Fragen in dem Test zu bestimmen, und darin verschachtelt eine dritte `for`-Schleife, um die Multiple-Choice-Antworten für die einzelnen Fragen zu generiert. Nach diesem Schritt sieht der Code wie folgt aus:

```
#! python3
# randomQuizGenerator.py - Erstellt Testfragebogen mit Fragen und Antworten
# in zufälliger Reihenfolge sowie die zugehörigen Lösungsbogen

-- schnipp --

# Durchläuft alle 50 Staaten und erstellt eine Frage für jeden
for questionNum in range(50):

    # Ruft die richtigen und falschen Antworten ab
    correctAnswer = capitals[states[questionNum]]      ❶
    wrongAnswers = list(capitals.values())      ❷
    del wrongAnswers[wrongAnswers.index(correctAnswer)] ❸
    wrongAnswers = random.sample(wrongAnswers, 3)      ❹
    answerOptions = wrongAnswers + [correctAnswer]      ❺
    random.shuffle(answerOptions)      ❻

    # TODO: Fragen und mögliche Antworten in die Testdatei schreiben

    # TODO: Lösungsschlüssel in eine Datei schreiben
```

An die richtige Antwort kommen wir ganz leicht, denn sie ist als Wert im Dictionary `capitals` gespeichert (❶). Die Schleife durchläuft die Bundesstaaten in der durcheinandergewürfelten Liste `states` von `states[0]` bis `states[49]`, ruft die einzelnen Staaten in `capitals` ab und speichert die zugehörige Hauptstadt in `correctAnswer`.

Die Liste der angebotenen falschen Antworten zusammenzustellen, ist dagegen etwas kniffliger. Dazu können Sie *sämtliche* Werte aus dem Dictionary `capitals` kopieren (❷), die richtige Antwort entfernen (❸) und anschließend drei zufällige Werte aus dieser Liste auswählen (❹). Diese zufällige Auswahl lässt sich mit `random.sample()` ganz leicht treffen. Das erste Argument dieser Funktion ist die Liste, aus der Sie auswählen wollen, das zweite die Anzahl der gewünschten Werte. Die

komplette Liste der angebotenen Antwortmöglichkeiten besteht aus diesen drei falschen und der richtigen Antwort (❸). Als Letztes werden diese Antworten zufällig geordnet, damit die richtige Antwort nicht immer D ist.

#### Schritt 4: Den Inhalt der Dateien für die Frage- und Lösungsbogen schreiben

Jetzt müssen wir nur noch die Fragen in die Fragebogen und die richtigen Antworten in die Lösungsbogen schreiben. Ergänzen Sie den Code wie folgt:

```
#! python3
# randomQuizGenerator.py - Erstellt Testfragebogen mit Fragen und Antworten
# in zufälliger Reihenfolge sowie die zugehörigen Lösungsbogen

-- schnipp --

# Durchläuft alle 50 Staaten und erstellt eine Frage für jeden
for questionNum in range(50):

-- schnipp --

# Write the question and the answer options to the quiz file.
quizFile.write('%s. What is the capital of %s?\n' % (questionNum + 1,
    states[questionNum]))
for i in range(4): ❶
    quizFile.write('    %s. %s\n' % ('ABCD'[i], answerOptions[i])) ❷
    quizFile.write('\n')

# Write the answer key to a file.
answerKeyFile.write('%s. %s\n' % (questionNum + 1, 'ABCD'[ ❸
    answerOptions.index(correctAnswer)]))

quizFile.close()
answerKeyFile.close()
```

Eine Schleife, die die Integerzahlen von 0 bis 3 durchläuft, schreibt die Antwortmöglichkeiten in die Liste `answerOptions` (❶). Der Ausdruck `'ABCD'[i]` bei (❷) behandelt den String `'ABCD'` als Array und wird in den aufeinanderfolgenden Schleifeniterationen nacheinander zu `'A'`, `'B'`, `'C'` und schließlich `'D'` ausgewertet.

In der letzten Zeile (❸) sucht der Ausdruck `answerOptions.index(correctAnswer)` den Integerindex der zufällig geordneten Antwortoptionen. Der Ausdruck `'ABCD'[answerOptions.index(correctAnswer)]` ergibt den Kennbuchstaben der richtigen Antwort, der dann in die Lösungsdatei geschrieben wird.

Die Datei `capitalquiz1.txt`, die dieses Programm ausgibt, sieht ähnlich aus wie im folgenden Beispiel, wobei die Reihenfolge der Fragen und die angebotenen Antworten natürlich von Fall zu Fall abweichen, da sie vom Ergebnis der Aufrufe von `random.shuffle()` abhängen:

Name:

Date:

Period:

### State Capitals Quiz (Form 1)

1. What is the capital of West Virginia?

- A. Hartford
- B. Santa Fe
- C. Harrisburg
- D. Charleston

2. What is the capital of Colorado?

- A. Raleigh
- B. Harrisburg
- C. Denver
- D. Lincoln

-- schnipp --

Die zugehörige Lösungsdatei *capitalsquiz\_answers1.txt* sieht wie folgt aus:

- 1. D
- 2. C
- 3. A
- 4. C

--snip--

## Projekt: Mehrfach-Zwischenablage

Stellen Sie sich vor, Sie haben die langweilige Aufgabe, auf einer Webseite oder in einer Software viele Formulare mit mehreren Textfeldern auszufüllen. Dank der Zwischenablage müssen Sie ein und denselben Text nicht immer wieder neu eingeben. Allerdings kann sich in der Zwischenablage immer nur ein Text auf einmal befinden. Wenn Sie mehrere verschiedene Texte kopieren und einfügen müssen, bleibt Ihnen trotzdem nichts anderes übrig, als immer wieder die gleichen Texte zu markieren und zu kopieren.

Allerdings können Sie ein Python-Programm schreiben, das sich mehrere Texte merken kann. Diese »Mehrfach-Zwischenablage« werden wir *mcb.pyw* nennen (wobei *mcb* für *multiclipboard* steht). Die Erweiterung *.pyw* bedeutet, dass Python bei der Ausführung dieses Programms kein Terminal-Fenster anzeigt. (Mehr darüber erfahren Sie in Anhang B.)

Das Programm legt jeden einzelnen in die Zwischenablage kopierten Text unter einem Schlüsselwort ab. Wenn Sie beispielsweise `py mcb.pyw save spam` ausführen, wird der aktuelle Inhalt der Zwischenablage unter dem Schlüsselwort `spam` gespeichert. Diesen Text können Sie dann später mit `py mcb.pyw spam` wieder in die Zwischenablage laden. Wenn Sie zwischendurch vergessen sollten, welche Schlüsselwörter Sie verwendet haben, können Sie mit `py mcb.pyw list` eine Liste aller Schlüsselwörter in die Zwischenablage kopieren.

Das Programm soll folgende Aufgaben erledigen:

- Die Befehlszeilenargumente untersuchen
- Bei dem Argument `save` den Inhalt der Schlüsselablage unter dem angegebenen Schlüsselwort speichern
- Bei dem Argument `list` alle Schlüsselwörter in die Zwischenablage kopieren
- Andernfalls den unter dem übergebenen Schlüsselwort gespeicherten Text in die Zwischenablage übertragen

Der Code muss daher Folgendes tun:

- Die Befehlszeilenargumente in `sys.argv` lesen
- In der Zwischenablage lesen und schreiben
- Eine Shelf-Datei speichern und laden

Unter Windows können Sie das Skript ganz einfach über das Fenster *Ausführen* starten, wenn Sie die Batchdatei `mcb.bat` mit dem folgenden Inhalt erstellen:

```
@pyw.exe C:\Python34\mcb.pyw %*
```

### Schritt 1: Kommentare und Vorbereitungen für die Shelf-Daten

Als Erstes erstellen wir ein Skelett für das Skript, das einige Kommentare enthält und grundlegende Aufgaben zur Einrichtung erfüllt. Der Code sieht vorläufig wie folgt aus:

```
#! python3
# mcb.pyw - Speichert Text und lädt ihn in die Zwischenablage
# Nutzung: py.exe mcb.pyw save <Schlüssel> - Speichert den Inhalt der
# Zwischenablage unter dem ❶
# Schlüssel
#       py.exe mcb.pyw <Schlüssel> - Lädt den Wert zu dem Schlüssel in
#       die Zwischenablage
#       py.exe mcb.pyw list - Lädt alle Schlüsselwörter in die
#       Zwischenablage
```

```

import shelve, pyperclip, sys ❷

mcbShelf = shelve.open('mcb') ❸

# TODO: Inhalt der Zwischenablage speichern

# TODO: Schlüsselwörter auflisten und Inhalt laden

mcbShelf.close()

```

Es ist gängige Praxis, am Anfang einer Skriptdatei Kommentare mit allgemeinen Hinweisen zur Nutzung anzugeben (❶). Sollten Sie vergessen, wie Sie das Skript ausführen müssen, können Sie immer auf diese Gedächtnisstütze zurückgreifen. Nach den Kommentaren importieren Sie die benötigten Module (❷). Für das Kopieren und Einfügen mithilfe der Zwischenablage brauchen Sie das Modul `pyperclip` und zum Lesen der Befehlszeilenargumente ist das Modul `sys` erforderlich. Auch das `shelve`-Modul wird benötigt, denn wenn der Benutzer einen neuen Zwischenablagetext speichern möchte, so geschieht das mithilfe einer Shelf-Datei. Soll der Text später wieder in die Zwischenablage zurückkopiert werden, öffnen Sie einfach die Shelf-Datei und laden den Inhalt in das Programm. Die Namen der Shelf-Dateien erhalten das Präfix `mcb` (❸).

## Schritt 2: Den Inhalt der Zwischenablage unter einem Schlüsselwort speichern

Je nachdem, ob der Benutzer Text unter einem Schlüsselwort speichern, Text in die Zwischenablage laden oder alle vorhandenen Schlüsselwörter auflisten möchte, führt das Programm verschiedene Aufgaben aus. Beginnen wir mit der ersten dieser Aufgaben. Ergänzen Sie den Code wie folgt:

```

#! python3
# mcb.pyw - Speichert Text und lädt ihn in die Zwischenablage
-- schnipp --

# Speichert den Inhalt der Zwischenablage
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save': ❶
    mcbShelf[sys.argv[2]] = pyperclip.paste() ❷
elif len(sys.argv) == 2:
    # TODO: Schlüsselwörter auflisten und Inhalt laden ❸

mcbShelf.close()

```

Wenn das erste Befehlszeilenargument (das in der Liste `sys.argv` immer am Index 1 steht) 'save' lautet (❶), dann ist das zweite Argument das Schlüsselwort, unter dem der aktuelle Inhalt der Zwischenablage gespeichert werden soll. Dieses Schlüs-

selwort wird als Schlüssel für `mcbShelf` verwendet und der Wert ist der Text, der sich zurzeit in der Zwischenablage befindet (❷).

Gibt es nur ein Befehlszeilenargument, dann handelt es sich dabei entweder um 'list' oder um ein Schlüsselwort, dessen zugehöriger Wert in die Zwischenablage geladen werden soll. Diesen Code werden Sie später schreiben. Vorläufig steht hier nur der TODO-Kommentar als Platzhalter und Erinnerungsstütze (❸).

### Schritt 3: Schlüsselwörter auflisten und Inhalte laden

Implementieren wir nun die restlichen beiden Fälle, in denen der Benutzer über ein Schlüsselwort Text in die Zwischenablage laden oder die Liste der verfügbaren Schlüsselwörter abrufen möchte. Ergänzen Sie den Code wie folgt:

```
#! python3
# mcb.pyw - Speichert Text und lädt ihn in die Zwischenablage
-- schnipp --

# Speichert den Inhalt der Zwischenablage
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':    1
    mcbShelf[sys.argv[2]] = pyperclip.paste()    2
elif len(sys.argv) == 2:
    # Listet Schlüsselwörter auf und lädt Inhalte
    if sys.argv[1].lower() == 'list':    ❶
        pyperclip.copy(str(list(mcbShelf.keys())))    ❷
    elif sys.argv[1] in mcbShelf:
        pyperclip.copy(mcbShelf[sys.argv[1]])    ❸

mcbShelf.close()
```

Wenn es nur ein Befehlszeilenargument gibt, prüfen wir zunächst, ob es 'list' ist (❶). Ist das tatsächlich der Fall, wird eine Stringdarstellung der Liste von Shelf-Schlüsseln in die Zwischenablage kopiert (❷). Der Benutzer kann diese Liste dann in einen Texteditor kopieren, um sie zu lesen.

Andernfalls können Sie davon ausgehen, dass es sich bei dem Befehlszeilenargument um ein Schlüsselwort handelt. Wenn es in `mcbShelf` als Schlüssel vorkommt, laden Sie den zugehörigen Wert in die Zwischenablage (❸).

Das war es auch schon! Je nachdem, welches Betriebssystem auf Ihrem Computer läuft, sind zum Starten dieses Programms unterschiedliche Schritte erforderlich. Weitere Hinweise zu den verschiedenen Betriebssystemen erhalten Sie in Anhang B.

In dem Passwortmanager aus Kapitel 6 haben Sie Passwörter in einem Dictionary gespeichert. Um Passwörter hinzuzufügen oder zu ändern, mussten Sie den Quellcode des Programms ändern. Das ist jedoch keine gute Vorgehensweise, da

sich durchschnittliche Benutzer nicht trauen, in den Quellcode einzugreifen, nur um Daten zu ändern. Außerdem laufen Sie bei jeder Änderung am Quellcode Gefahr, versehentlich irgendwelche Bugs hervorzurufen. Wenn Sie die Daten für ein Programm statt im Code an anderer Stelle speichern, lässt sich das Programm von anderen einfacher benutzen und ist weniger fehleranfällig.

## Zusammenfassung

Dateien werden in Ordnern (Verzeichnissen) abgelegt, wobei der Pfad den Speicherort der Datei angibt. Da jedes Programm, das auf einem Computer läuft, über ein Arbeitsverzeichnis verfügt, können Dateipfade relativ zu diesem aktuellen Speicherort angegeben werden, anstatt jedes Mal den vollständigen (absoluten) Pfad auszuschreiben. Das Modul `os.path` verfügt über viele Funktionen für den Umgang mit Dateipfaden.

Programme können auch direkt mit den Inhalten von Textdateien umgehen. Mit der Funktion `open()` öffnen Sie solche Dateien, um ihren Inhalt zu lesen, entweder mit `read()` in Form eines einzigen, langen Strings oder mit `readlines()` als Liste von Strings. Die Funktion `open()` kann Dateien auch im Schreib- oder Anhangemode öffnen, um neue Textdateien zu erstellen oder Inhalte zu vorhandenen Textdateien hinzuzufügen.

In den vorherigen Kapiteln haben Sie die Zwischenablage verwendet, um dem Programm große Mengen von Texten zu übergeben, anstatt den ganzen Text manuell einzugeben. Wie Sie jetzt wissen, können Ihre Programme Dateien auch direkt von der Festplatte lesen. Das ist ein großer Vorteil, da solche Dateien nicht so flüchtig sind wie der Inhalt der Zwischenablage.

Im nächsten Kapitel lernen Sie den Umgang mit Dateien – wie Sie sie kopieren, löschen, umbenennen, verschieben usw.

## Wiederholungsfragen

1. Wozu ist ein relativer Pfad relativ?
2. Womit beginnt ein absoluter Pfad?
3. Was machen die Funktionen `os.getcwd()` und `os.chdir()`?
4. Worum handelt es sich bei den Ordnern . und .. ?
5. Welcher Teil von `C:\bacon\eggs\spam.txt` ist der Verzeichnisname und welcher der Grundname?
6. Welche drei Modusargumente können Sie an die Funktion `open()` übergeben?
7. Was geschieht, wenn Sie eine bereits vorhandene Datei im Schreibmodus öffnen?

8. Was ist der Unterschied zwischen den Methoden `read()` und `readlines()`?
9. Welche Datenstruktur stellt ein Shelf-Wert dar?

## **Übungsprojekte**

Entwerfen und schreiben Sie zur Übung die folgenden Programme:

### **Erweiterte Mehrfach-Zwischenablage**

Ergänzen Sie das Programm für die Mehrfach-Zwischenablage aus diesem Kapitel um das Befehlszeilenargument `delete <schlüsselwort>`, mit dem Sie ein Schlüsselwort aus dem Shelf löschen. Fügen Sie außerdem das Befehlszeilenargument `delete` hinzu, das *alle* Schlüsselwörter löscht.

### **Lückentextspiel**

Erstellen Sie ein Programm für ein Lückentextspiel, bei dem der Benutzer an den Stellen, an denen in der Textdatei *ADJECTIVE*, *NOUN*, *ADVERB* oder *VERB* vorkommt, ein eigenes Wort der entsprechenden Art eingeben kann. Die Textdatei kann dabei beispielsweise wie folgt aussehen:

```
The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN was  
unaffected by these events.
```

Das Programm findet die zu ersetzenen Stellen und fordert den Benutzer auf, Wörter dafür einzugeben:

```
Enter an adjective:  
silly  
Enter a noun:  
chandelier  
Enter a verb:  
screamed  
Enter a noun:  
pickup truck
```

Dadurch entsteht folgender Text:

```
The silly panda walked to the chandelier and then screamed. A nearby pickup  
truck was unaffected by these events.
```

Das Ergebnis soll auf dem Bildschirm ausgegeben und in einer neuen Textdatei gespeichert werden.

### Regex-Suche

Schreiben Sie ein Programm, das alle *.txt*-Dateien in einem Ordner öffnet und nach sämtlichen Zeilen sucht, die mit dem vom Benutzer angegebenen regulären Ausdruck übereinstimmen. Die Ergebnisse sollen auf dem Bildschirm ausgegeben werden.

# 9

## Dateien verwalten



Im vorherigen Kapitel haben Sie gelernt, wie Sie in Python neue Dateien erstellen und darin schreiben können. Allerdings können Ihre Programme auch die bereits vorhandenen Dateien auf Ihrer Festplatte verwalten. Vielleicht haben Sie schon einmal die Erfahrung machen müssen, wie es ist, Dutzende, Hunderte oder gar Tausende von Dateien in einem Ordner manuell kopieren, umbenennen, verschieben oder komprimieren zu müssen. Denken Sie auch an Aufgaben wie die folgenden:

- Sämtliche PDF-Dateien (und zwar *nur* die PDF-Dateien) in allen Unterordnern eines Ordners kopieren
- Führende Nullen in den Namen sämtlicher Dateien eines Ordners entfernen, in denen sich Hunderte von Dateien mit Namen wie *spam001.txt*, *spam002.txt* usw. befinden
- Die Inhalte mehrerer Ordner in einer ZIP-Datei komprimieren (was als einfache Backup-Lösung dienen könnte)

All diese langweiligen Aufgaben schreien geradezu danach, in Python automatisiert zu werden. Wenn Sie Ihren Computer so programmieren, dass er diese Aufgaben für Sie erledigt, haben Sie einen fixen Bürohelfer, der niemals Fehler macht.

Bei der Arbeit mit Dateien ist es sehr hilfreich, die Dateinamenerweiterung (.txt, .pdf, .jpg usw.) sofort erkennen zu können. In OS X und Linux werden diese Endungen im Dateibrowser gewöhnlich automatisch angezeigt. Bei Windows hingegen werden Standardendungen normalerweise unterschlagen. Um sie einzublenden, wählen Sie *Start > Systemsteuerung > Darstellung und Anpassung > Ordneroptionen*. Entfernen Sie auf der Registerkarte *Ansicht* unter *Erweiterte Einstellungen* das Häkchen bei *Erweiterungen bei bekannten Dateitypen ausblenden*. (Bei anderen Versionen als Windows 7 kann der Weg zum Dialogfeld *Ordneroptionen* von der hier beschriebenen Vorgehensweise abweichen.)

## Das Modul shutil

Das Modul `shutil` (*Shell Utilities*) enthält Funktionen, mit denen Sie Dateien in Ihren Python-Programmen kopieren, verschieben, umbenennen und löschen können. Um sie nutzen zu können, müssen Sie `shutil` zunächst importieren.

### Dateien und Ordner kopieren

Im Modul `shutil` gibt es Funktionen, um einzelne Dateien und ganze Ordner zu kopieren.

Die Funktion `shutil.copy(source, destination)` kopiert die Datei `source` in den Ordner mit dem Pfad `destination` (wobei sowohl `source` als auch `destination` Strings sein müssen). Wenn `destination` ein Dateiname ist, erhält die kopierte Datei diesen neuen Namen. Die Funktion gibt einen String mit dem Pfad zu der kopierten Datei zurück.

Um `shutil.copy()` in Aktion zu erleben, geben Sie Folgendes in die interaktive Shell ein:

```
>>> import shutil, os
>>> os.chdir('C:\\')
>>> shutil.copy('C:\\spam.txt', 'C:\\delicious')   ❶
'C:\\delicious\\spam.txt'
>>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt') ❷
'C:\\delicious\\eggs2.txt'
```

Der erste Aufruf von `shutil.copy()` kopiert die Quelldatei `C:\spam.txt` in den Ordner `C:\delicious`. Der Rückgabewert ist der Pfad der neuen Datei. Da als Ziel ein Ordner angegeben ist (❶), wird für die neue Datei der ursprüngliche Dateiname `spam.txt` verwendet. Beim zweiten Aufruf von `shutil.copy()` wird die Datei `C:\`

*eggs.txt* ebenfalls in den Ordner C:\delicious kopiert, wobei die Kopie jedoch den Namen *eggs2.txt* erhält.

Mit `shutil.copy()` können Sie nur eine einzelne Datei kopieren. Dagegen kopiert `shutil.copytree(source, destination)` den gesamten Ordner im Pfad *source* einschließlich aller darin enthaltenen Unterordner und Dateien in den Ordner im Pfad *destination*. Auch hier sind die Parameter *source* und *destination* Strings. Die Funktion gibt den Pfad der Ordnerkopie zurück.

Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import shutil, os  
>>> os.chdir('C:\\\\')  
>>> shutil.copytree('C:\\\\bacon', 'C:\\\\bacon_backup')  
'C:\\\\bacon_backup'
```

Hier wird durch den Aufruf von `shutil.copytree()` ein neuer Ordner namens *bacon\_backup* erstellt, der denselben Inhalt hat wie der ursprüngliche Ordner *bacon*. Damit haben Sie eine Sicherheitskopie Ihrer kostbaren Daten angelegt.

### Dateien und Ordner verschieben und umbenennen

Mit `shutil.move(source, destination)` verschieben Sie die Datei oder den Ordner im Pfad *source* zum Pfad *destination*, wobei ein String mit dem absoluten Pfad zum neuen Speicherort zurückgegeben wird.

Wenn *destination* auf einen Ordner verweist, wird die Quelldatei in diesen Zielordner verschoben, behält aber ihren Namen bei, wie das folgende Beispiel zeigt:

```
>>> import shutil  
>>> shutil.move('C:\\\\bacon.txt', 'C:\\\\eggs')  
'C:\\\\eggs\\\\bacon.txt'
```

Unter der Voraussetzung, dass es im Verzeichnis C:\ bereits einen Ordner namens *eggs* gibt, besagt dieser Aufruf von `shutil.move()`: »Verschiebe C:\bacon.txt in den Ordner C:\eggs.«

Gibt es in C:\eggs bereits eine Datei namens *bacon.txt*, so wird sie überschrieben. Da so etwas sehr leicht versehentlich geschehen kann, müssen Sie bei der Verwendung von `move()` sehr vorsichtig vorgehen.

Im Zielpfad können Sie auch einen Dateinamen angeben. Im folgenden Beispiel wird die Quelldatei nicht nur verschoben, sondern auch umbenannt:

```
>>> shutil.move('C:\\\\bacon.txt', 'C:\\\\eggs\\\\new_bacon.txt')  
'C:\\\\eggs\\\\new_bacon.txt'
```

Diese Anweisung bedeutet: »Verschiebe die Datei C:\bacon.txt in den Ordner C:\eggs und benenne dabei gleich in new\_bacon.txt um.«

Bei den beiden vorstehenden Beispielen sind wir davon ausgegangen, dass es im Verzeichnis C:\ bereits einen Ordner namens *eggs* gibt. Ist das nicht der Fall, benennt `move()` die Datei *bacon.txt* in *eggs* um:

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')  
'C:\\eggs'
```

Da die Funktion `move()` hier keinen Ordner namens *eggs* im Verzeichnis C:\ finden kann, geht sie davon aus, dass der Parameter *destination* einen Dateinamen und keinen Ordner angibt, und benennt *bacon.txt* daher in *eggs* um (also in eine Textdatei ohne die Endung *.txt*), was höchstwahrscheinlich nicht das ist, was Sie tun wollten. Solche Fehler lassen sich in einem Programm oft nur schwer finden, da `move()` völlig unauffällig ganz andere Dinge tun kann, als Sie erwarten. Das ist ein weiterer Grund, um bei der Verwendung von `move()` Vorsicht walten zu lassen.

Die Ordner, aus denen sich der Zielpfad zusammensetzt, müssen vorhanden sein, da Python sonst eine Ausnahme auflöst. Das können Sie in der interaktiven Shell wie folgt ausprobieren:

```
>>> shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')  
Traceback (most recent call last):  
  File "C:\\Python34\\lib\\shutil.py", line 521, in move  
    os.rename(src, real_dst)  
FileNotFoundError: [WinError 3] The system cannot find the path specified:  
'spam.txt' -> 'c:\\does_not_exist\\eggs\\ham'
```

During handling of the above exception, another exception occurred:

```
Traceback (most recent call last):  
  File "<pyshell#29>", line 1, in <module>  
    shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')  
  File "C:\\Python34\\lib\\shutil.py", line 533, in move  
    copy2(src, real_dst)  
  File "C:\\Python34\\lib\\shutil.py", line 244, in copy2  
    copyfile(src, dst, follow_symlinks=follow_symlinks)  
  File "C:\\Python34\\lib\\shutil.py", line 108, in copyfile  
    with open(dst, 'wb') as fdst:  
FileNotFoundError: [Errno 2] No such file or directory: 'c:\\does_not_exist\\eggs\\ham'
```

Python sucht nach den Unterordnern *ham* und *eggs* in dem Verzeichnis *does\_not\_exist*, das aber gar nicht vorhanden ist, weshalb *spam.txt* auch nicht in den angegebenen Pfad verschoben werden kann.

## Dateien und Ordner unwiederbringlich löschen

Mit den Funktionen im Modul `os` können Sie einzelne Dateien und einzelne leere Ordner löschen. Wollen Sie hingegen einen Ordner samt seiner Inhalte entfernen, müssen Sie das Modul `shutil` verwenden:

- Die Funktion `os.unlink(path)` löscht die Datei im angegebenen Pfad.
- Die Funktion `os.rmdir(path)` löscht den Ordner im angegebenen Pfad. In diesem Ordner dürfen sich keine Dateien oder Ordner mehr befinden.
- Die Funktion `os.rmtree(path)` löscht den Ordner im angegebenen Pfad samt aller darin enthaltenen Dateien und Ordner.

Seien Sie sehr vorsichtig, wenn Sie diese Funktionen in Ihren Programmen einsetzen! Es ist eine gute Idee, zunächst einen Testlauf des Programms durchzuführen, bei dem die Aufrufe auskommentiert sind und zusätzliche `print()`-Aufrufe verwendet werden, um anzuzeigen, welche Dateien gelöscht werden. Das folgende Python-Programm soll eigentlich alle Dateien mit der Erweiterung `.txt` löschen, entfernt aufgrund eines Tippfehlers (hier fett hervorgehoben) jedoch alle `.rxt`-Dateien:

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        os.unlink(filename)
```

Wenn Sie über eine wichtige Datei mit der Endung `.rxt` verfügen, würde sie durch dieses Programm versehentlich unwiederbringlich gelöscht. Führen Sie daher zunächst einen Testlauf in folgender Form durch:

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        #os.unlink(filename)
        print(filename)
```

Da der Aufruf von `os.unlink()` auskommentiert ist, ignoriert Python ihn. Stattdessen werden nur die Namen der Dateien ausgegeben, die mit diesem Befehl gelöscht werden würden. Bei diesem Testlauf können Sie also sofort erkennen, dass das Programm `.rxt`- statt `.txt`-Dateien entfernt.

Wenn Sie sich vergewissert haben, dass das Programm wie beabsichtigt funktioniert, entfernen Sie die Zeile `print(filename)` und das Kommentarzeichen vor `os.unlink(filename)`. Führen Sie dann das Programm aus, um die Dateien tatsächlich zu löschen.

## Sicheres Löschen mit dem Modul send2trash

Da die integrierte Python-Funktion `shutil.rmtree()` Dateien und Ordner unweiterbringlich löscht, kann es ziemlich gefährlich sein, sie zu nutzen. Eine bessere Möglichkeit, um Dateien und Ordner zu entfernen, bietet das Drittanbietermodul `send2trash`. Um es zu installieren, führen Sie `pip install send2trash` in einem Terminal-Fenster aus. (Eine ausführliche Erklärung zur Installation von Drittanbietermodulen erhalten Sie in Anhang A.)

Die Verwendung von `send2trash` ist viel sicherer als die regulären Löschfunktionen von Python, da es die betroffenen Ordner und Dateien in den Papierkorb verschiebt, anstatt sie gleich ein für allemal zu entfernen. Wenn `send2trash` aufgrund eines Fehlers in Ihrem Programm versehentlich etwas löscht, was Sie noch behalten wollten, können Sie es aus dem Papierkorb wiederherstellen.

Probieren Sie nach der Installation von `send2trash` Folgendes in der interaktiven Shell aus:

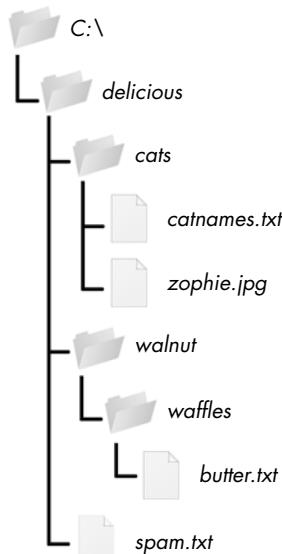
```
>>> import send2trash
>>> baconFile = open('bacon.txt', 'a')      # Erstellt die Datei
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> send2trash.send2trash('bacon.txt')
```

Im Allgemeinen sollten Sie zum Löschen von Dateien und Ordnern bevorzugt `send2trash.send2trash()` verwenden. Dadurch haben Sie die Möglichkeit, die Dateien später wiederherzustellen. Allerdings können Sie damit keinen Festplattenplatz freigeben, wie es eine dauerhafte Löschung ermöglicht. Wenn es darauf ankommt, Platz zu gewinnen, nehmen Sie die `os-` und `shutil-`Funktionen. Beachten Sie, dass Sie Dateien mit `send2trash()` nur in den Papierkorb verschieben, aber nicht daraus wiederherstellen können.

## Einen Verzeichnisbaum durchlaufen

Nehmen wir an, Sie möchten sämtliche Dateien in einem Ordner und in allen seinen Unterordnern umbenennen. Dazu müssen Sie den gesamten Verzeichnisbaum durchlaufen und alle Dateien bearbeiten, die Sie dabei finden. Es kann ziemlich knifflig sein, ein Programm zu schreiben, das genau das tut, aber zum Glück gibt es in Python eine Funktion, die diese Aufgabe für Sie erledigt.

Schauen Sie sich in Abb. 9–1 die Struktur des Ordners `C:\delicious` und seiner Inhalte an.



**Abb. 9–1** Dieser Beispielordner enthält drei Unterordner und insgesamt vier Dateien.

Das folgende Beispielprogramm wendet die Funktion `os.walk()` auf den Verzeichnisbaum aus Abb. 9–1 an:

```
import os

for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('The current folder is ' + folderName)

    for subfolder in subfolders:
        print('SUBFOLDER OF ' + folderName + ': ' + subfolder)
    for filename in filenames:
        print('FILE INSIDE ' + folderName + ': ' + filename)
    print('')
```

Der Funktion `os.walk()` wird ein einziger Stringwert übergeben, nämlich der Pfad eines Ordners. Wenn Sie diese Funktion in einer `for`-Schleife verwenden, wird der Verzeichnisbaum durchlaufen – ähnlich wie die Funktion `range()` bewirkt, dass die Schleife einen Zahnbereich durchläuft. Im Gegensatz zu `range()` gibt `os.walk()` jedoch bei jeder Iteration drei Werte zurück:

1. Einen String mit dem Namen des aktuellen Ordners
2. Eine Liste von Strings mit den Namen der Unterordner im aktuellen Ordner
3. Eine Liste von Strings mit den Namen der Dateien im aktuellen Ordner

Der »aktuelle Ordner« ist hier der Ordner, der in der vorliegenden Iteration der `for`-Schleife gerade durchlaufen wird. Das aktuelle Arbeitsverzeichnis des Programms wird durch `os.walk()` nicht geändert.

Ebenso, wie Sie im Code `i in range(10):` den Variablenamen `i` wählen können, ist es möglich, Variablenamen für die drei angeführten Werte zu vergeben. Gewöhnlich verwende ich `foldername`, `subfolders` und `filenames`.

Die Ausgabe dieses Programms sieht wie folgt aus:

```
The current folder is C:\delicious
SUBFOLDER OF C:\delicious: cats
SUBFOLDER OF C:\delicious: walnut
FILE INSIDE C:\delicious: spam.txt
```

```
The current folder is C:\delicious\cats
FILE INSIDE C:\delicious\cats: catnames.txt
FILE INSIDE C:\delicious\cats: zophie.jpg
```

```
The current folder is C:\delicious\walnut
SUBFOLDER OF C:\delicious\walnut: waffles
```

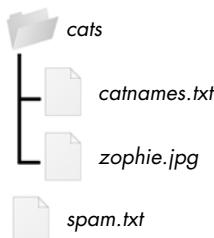
```
The current folder is C:\delicious\walnut\waffles
FILE INSIDE C:\delicious\walnut\waffles: butter.txt
```

Da `os.walk()` eine Liste der Strings für die Variablen `subfolder` und `filename` zurückgibt, können Sie diese Listen in eigenen `for`-Schleifen verwenden. Ersetzen Sie dazu die Aufrufe der Funktion `print()` durch Ihren eigenen Code. (Wenn Sie irgendeine dieser Variablen nicht brauchen, können Sie deren `for`-Schleife auch einfach entfernen.)

## Dateien mit der Methode `zipfile komprimieren`

Wahrscheinlich kennen Sie bereits ZIP-Dateien (mit der Endung `.zip`), die mehrere andere Dateien in komprimierter Form enthalten können. Durch diese Komprimierung wird die Dateigröße verringert, was insbesondere für die Übertragung über das Internet von Vorteil ist. Da ZIP-Dateien außerdem mehrere Dateien und Unterordner enthalten können, bilden sie eine praktische Möglichkeit, um mehrere Dateien in eine einzige zu packen. Ein solches *Archiv* kann dann auf einfache Weise z. B. an eine E-Mail angehängt werden.

Mit den Funktionen des Moduls `zipfile` können Sie Python-Programme schreiben, die in der Lage sind, ZIP-Dateien zu erstellen und zu öffnen (zu *entpacken*). In den folgenden Beispielen gehen wir davon aus, dass Sie eine ZIP-Datei namens `example.zip` mit dem Inhalt aus Abb. 9–2 haben.



**Abb. 9–2** Der Inhalt von *example.zip*

Diese ZIP-Datei können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen; Sie können die Beispiele aber auch mit einer anderen ZIP-Datei auf Ihrem Computer nachvollziehen.

### ZIP-Dateien lesen

Um den Inhalt einer ZIP-Datei zu lesen, müssen Sie zunächst ein ZipFile-Objekt (mit großem Z und großem F) erstellen. Im Prinzip ähneln diese Objekte den File-Objekten, die die Funktion open() zurückgibt: Auch bei ihnen handelt es sich um Werte, über die das Programm mit der Datei arbeitet. Um ein ZipFile-Objekt zu erstellen, rufen Sie die Funktion zipfile.ZipFile() auf und übergeben Ihr einen String mit dem Namen der gewünschten ZIP-Datei. Beachten Sie, dass der Name des Python-Moduls zipfile geschrieben wird, die Funktion aber ZipFile().

Geben Sie zur Veranschaulichung Folgendes in die interaktive Shell ein:#

```
>>> import zipfile, os
>>> os.chdir('C:\\')      # Wechselt zum Ordner mit example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
>>> 'Compressed file is %sx smaller!' % (round(spamInfo.file_size / spamInfo
.compress_size, 2))    ❶
'Compressed file is 3.63x smaller!'
>>> exampleZip.close()
```

ZipFile-Objekte verfügen über die Methode namelist(), die eine Liste der Strings mit den Namen aller Dateien und Ordner in der ZIP-Datei zurückgibt. Diese Strings können Sie an die ZipFile-Methode getinfo() übergeben, die ein ZipInfo-Objekt mit Informationen über die betreffende Datei zurückgibt. ZipInfo-Objekte verfügen

über ihre eigenen Attribute, z. B. `file_size` und `compress_size`, die Integerwerte für die ursprüngliche Dateigröße und die Größe der komprimierten Datei in Byte enthalten. Während ein `ZipFile`-Objekt für eine komplette Archivdatei steht, enthält ein `ZipInfo`-Objekt nur Informationen über eine *einzige Datei* in diesem Archiv.

Der Befehl bei (❶) berechnet, wie effizient `example.zip` komprimiert wurde. Dazu wird die ursprüngliche Dateigröße durch die Größe der komprimierten Version geteilt und das Ergebnis anstelle des Platzhalters `%s` in dem genannten String ausgegeben.

## ZIP-Dateien entpacken

Die Methode `extractall()` für `ZipFile`-Objekte entpackt alle Dateien und Ordner einer ZIP-Datei in das aktuelle Arbeitsverzeichnis.

```
>>> import zipfile, os
>>> os.chdir('C:\\')      # Wechselt zum Ordner mit example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.extractall() ❶
>>> exampleZip.close()
```

Bei der Ausführung dieses Codes wird der Inhalt von `example.zip` in C:\ entpackt. Wenn Sie nicht das Arbeitsverzeichnis verwenden wollen, können Sie `extractall()` optional auch den Namen des Ordners übergeben, in den die Dateien entpackt werden sollen. Falls dieser Ordner noch nicht vorhanden ist, wird er erstellt. Nehmen wir an, Sie ersetzen den Aufruf in (❶) durch `exampleZip.extractall('C:\\delicious')`. In diesem Fall werden die Dateien von `example.zip` in den neu erstellten Ordner C:\delicious entpackt.

Die `ZipFile`-Methode `extract()` entpackt nur eine einzige Datei der ZIP-Datei, wie das folgende Beispiel zeigt:

```
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
```

Der String, den Sie `extract()` übergeben, muss genau einem der Strings in der von `namelist()` zurückgegebenen Liste entsprechen. Optional können Sie als zweites Argument auch den Namen eines Ordners übergeben, in den die Datei statt in das Arbeitsverzeichnis entpackt werden soll. Wenn dieser Ordner nicht vorhanden ist, legt Python ihn an. Der von `extract()` zurückgegebene Wert ist der absolute Pfad der erstellten Datei.

### ZIP-Dateien erstellen und Inhalte hinzufügen

Um eigene ZIP-Dateien zu erstellen, öffnen Sie ein `ZipFile`-Objekt im *Schreibmodus*, indem Sie als zweites Argument '`w`' übergeben (ähnlich wie Sie der Funktion `open()` den String '`w`' übergeben, um eine Textdatei im Schreibmodus zu öffnen).

Wenn Sie der Methode `write()` eines `ZipFile`-Objekts einen Pfad übergeben, komprimiert Python die Dateien in diesem Pfad und fügt sie zu der ZIP-Datei hinzu. Das erste Argument von `write()` ist der String mit dem Namen der hinzuzufügenden Datei, das zweite ist der *Komprimierungstyp*, der dem Computer mitteilt, welchen Algorithmus er zum Komprimieren der Dateien verwenden soll. Für diesen Wert können Sie stets `zipfile.ZIP_DEFLATED` verwendet. (Dadurch wird der *Deflate*-Algorithmus festgelegt, der bei allen Datentypen gut funktioniert.) Proben Sie das wie folgt in der interaktiven Shell aus:

```
>>> import zipfile  
>>> newZip = zipfile.ZipFile('new.zip', 'w')  
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)  
>>> newZip.close()
```

Dieser Code erstellt die neue ZIP-Datei `new.zip`, die die Datei `spam.txt` in komprimierter Form enthält.

Wie bei Dateien werden die vorhandenen Inhalte einer ZIP-Datei im Schreibmodus gelöscht. Wenn Sie also einem vorhandenen ZIP-Archiv weitere Dateien hinzufügen möchten, müssen Sie '`a`' als zweites Argument an `zipfile.ZipFile()` übergeben, damit das Archiv im *Anhängemodus* geöffnet wird.

## Projekt: Amerikanische Datumsangaben in europäische ändern

Stellen Sie sich vor, Ihr Chef schickt Ihnen Tausende von Dateien, in deren Namen Datumsangaben im amerikanischen Format vorkommen (MM-DD-YYYY), und weist sie an, sie ins europäische Datumsformat (DD-MM-YYYY) umzuwandeln. Diese langweilige Aufgabe per Hand zu erledigen, könnte Sie einen ganzen Tag kosten! Schreiben wir daher ein Programm, das diese Aufgabe für Sie erledigt.

Das Programm muss Folgendes tun können:

- Alle Dateinamen im aktuellen Arbeitsverzeichnis nach amerikanischen Datumsangaben durchsuchen
- Bei allen Fundstellen die Tages- und die Monatsangabe im Dateinamen vertauschen

Der Code muss also folgende Aufgaben erfüllen:

- Einen regulären Ausdruck definieren, der das Textmuster amerikanischer Datumsangaben findet
- Alle Dateien im Arbeitsverzeichnis mithilfe von `os.listdir()` finden
- Alle Dateinamen durchlaufen und anhand des regulären Ausdrucks überprüfen, welche davon eine Datumsangabe enthalten
- Dateien mit Datumsangaben im Namen mithilfe von `shutil.move()` umbenennen

Für dieses Projekt öffnen Sie ein neues Dateieditorfenster und speichern den Code als `renameDates.py`.

### Schritt 1: Einen regulären Ausdruck für amerikanische Datumsangaben definieren

Im ersten Teil des Programms müssen Sie die erforderlichen Module importieren und einen regulären Ausdruck für Daten im Format MM-DD-YYYY definieren. Die TODO-Kommentare dienen als Gedächtnissstütze für die Teile des Programms, die Sie noch schreiben müssen. Wenn Sie solchen Kommentaren stets dieselbe Bezeichnung voranstellen wie hier TODO, können Sie sie später mit der Suchfunktion von IDLE ([Strg] + [F]) schnell finden. Der Code sieht in diesem Stadium wie folgt aus:

```
#! python3
# renameDates.py - Ändert amerikanische MM-DD-YYYY-Datumsangaben in
# Dateinamen in europäische DD-MM-YYYY-Datumsangaben

import shutil, os, re ①

# Regulärer Ausdruck für Dateinamen mit Datumsangaben im US-Format
datePattern = re.compile(r"""\^(.*?) # Gesamter Text vor dem Datum ②
    ((0|1)?\d)- # Ein oder zwei Ziffern für den Monat
    ((0|1|2|3)?\d)- # Ein oder zwei Ziffern für den Tag
    ((19|20)\d\d) # Vier Ziffern für das Jahr
    (.*)\$ # Gesamter Text nach dem Datum
    """, re.VERBOSE) ③

# TODO: Alle Dateien im Arbeitsverzeichnis durchlaufen
# TODO: Dateinamen ohne Datumsangabe überspringen
# TODO: Die einzelnen Teile des Dateinamens abrufen
# TODO: Dateinamen im europäischen Format zusammenstellen
# TODO: Den kompletten absoluten Pfad abrufen
# TODO: Dateien umbenennen
```

Wie Sie bereits wissen, können Sie die Funktion `shutil.move()` auch dazu verwenden, um Dateien umzubenennen. In diesem Fall sind die Argumente der ursprüng-

liche und der neue Dateiname. Da diese Funktion zum Modul `shutil` gehört, müssen Sie dieses zunächst importieren (❶).

Bevor Sie damit beginnen können, Dateien umzubenennen, müssen Sie jedoch zunächst einmal die betroffenen Dateien herausfinden. Geändert werden sollen nur Dateinamen, die Datumsangaben enthalten, z. B. `spam4-4-1984.txt` oder `01-03-2014eggs.zip`, wohingegen Dateien wie `littlebrother.epub` nicht angefasst werden.

Um das Datumsmuster zu finden, setzen Sie einen regulären Ausdruck ein. Nach dem Import des Moduls `re` rufen Sie `re.compile()` auf, um ein Regex-Objekt zu erstellen (❷). Die Übergabe von `re.VERBOSE` als zweites Argument (❸) ermöglicht es Ihnen, Weißraumzeichen und Kommentare in dem Ausdruck anzugeben, um ihn übersichtlicher zu gestalten.

Der reguläre Ausdruck beginnt mit `^(.*?)`, um beliebigen Text abzudecken, der in dem Dateinamen vor dem Datum steht. Die anschließende Gruppe `((0|1)?\d)` stellt die Monatsangabe dar. Die erste Ziffer kann 0 oder 1 sein, damit der reguläre Ausdruck sowohl 12 für Dezember als auch 02 für Februar findet. Allerdings ist diese erste Stelle optional, denn beispielsweise könnte der April sowohl mit 04 als auch einfach mit 4 angegeben sein. Die Gruppe für die Tagesangabe lautet `((0|1|2|3)?\d)` und folgt einer ähnlichen Logik, um beispielsweise 3, 03 und 31 als gültige Tagesangaben erkennen zu können. (Dieser reguläre Ausdruck akzeptiert auch einige ungültige Daten wie 4-31-2014, 2-29-2013 oder 0-15-2014. Bei Datumsangaben gibt es viele ziemlich knifflige Sonderfälle, die leicht übersehen werden. Der reguläre Ausdruck für dieses Beispielprogramm ist ziemlich einfach gehalten, funktioniert im Rahmen der gestellten Aufgabe aber ausreichend gut.)

1885 ist zwar eine gültige Jahresangabe, aber Sie können die Suche auch auf Jahre im 20. und 21. Jahrhundert einschränken, um zu verhindern, dass das Programm versehentlich Dateinamen mit datumsähnlichen Zahlenangaben umbenamt, die gar keine Kalenderdaten sind, z. B. `10-10-1000.txt`.

Der letzte Teil des regulären Ausdrucks, `(.*?)$`, deckt jeglichen Text ab, der hinter dem Datum steht.

## Schritt 2: Die einzelnen Teile der Datumsangabe in den Dateinamen ermitteln

Als Nächstes muss das Programm die von `os.listdir()` zurückgegebene Liste der Dateinamenstrings in einer Schleife durchlaufen und die einzelnen Namen mit dem regulären Ausdruck vergleichen. Alle Dateien, deren Namen keine Datumsangabe enthalten, sollen übersprungen werden. Bei Dateien mit einer Datumsangabe im Namen wird der gefundene Text in mehreren Variablen gespeichert. Ersetzen Sie die ersten drei TODO-Kommentare in Ihrem Programm durch den folgenden Code:

```

#! python3
# renameDates.py - Ändert amerikanische MM-DD-YYYY-Datumsangaben in
# Dateinamen in europäische DD-MM-YYYY-Datumsangaben

-- schnipp --

# Durchläuft die Dateien im Arbeitsverzeichnis
for amerFilename in os.listdir('.'):
    mo = datePattern.search(amerFilename)

    # Überspringt Dateien ohne Datumsangabe
    if mo == None: ①
        continue ②

    # Ruft die einzelnen Teile des Dateinamens ab ③
    beforePart = mo.group(1)
    monthPart  = mo.group(2)
    dayPart    = mo.group(4)
    yearPart   = mo.group(6)
    afterPart  = mo.group(8)

-- schnipp --

```

Wenn das von `search()` zurückgegebene Match-Objekt den Wert `None` hat (①), dann stimmt der Dateiname in `amerFilename` nicht mit dem regulären Ausdruck überein. Die Anweisung `continue` (②) sorgt dafür, dass der Rest der Schleife übersprungen wird und das Programm mit dem nächsten Dateinamen fortfährt.

Anderenfalls werden die einzelnen Strings, die den verschiedenen Gruppen des regulären Ausdrucks entsprechen, in den Variablen `beforePart`, `monthPart`, `dayPart`, `yearPart` und `afterPart` gespeichert (③). Aus den Strings in diesen Variablen werden im nächsten Schritt die Dateinamen mit den europäischen Datumsangaben zusammengestellt.

Um mit den Gruppennummern nicht durcheinanderzukommen, lesen Sie den regulären Ausdruck von links nach rechts und erhöhen Sie die Nummer jedes Mal um 1, wenn Sie auf eine öffnende Klammer stoßen. Ein großes Gerüst des Ausdrucks ohne den Code im Einzelnen kann Ihnen helfen, sich die Gruppen besser vor Augen zu halten:

```

datePattern = re.compile(r"""\^(1) # Gesamter Text vor dem Datum ②
(2 (3))- # Ein oder zwei Ziffern für den Monat
(4 (5))-- # Ein oder zwei Ziffern für den Tag
(6 (7))- # Vier Ziffern für das Jahr
(8)$ # Gesamter Text nach dem Datum
""", re.VERBOSE)

```

Hier stehen die Zahlen 1 bis 8 für die Gruppen in dem regulären Ausdruck. Eine solche Übersicht, die nur die Klammern und Gruppennummern enthält, vermittelt Ihnen ein besseres Verständnis des Ausdrucks für den weiteren Verlauf des Programms.

### Schritt 3: Die neuen Dateinamen zusammenstellen und die Dateien umbenennen

Im letzten Schritt verketten Sie die Strings in den Variablen, die im vorherigen Schritt erstellt wurden, zu Dateinamen mit europäischer Datumsangabe, also mit der Tagesangabe vor der Monatsangabe. Ersetzen Sie die drei restlichen TODO-Kommentare in dem Programm durch folgenden Code:

```
#! python3
# renameDates.py - Ändert amerikanische MM-DD-YYYY-Datumsangaben in
# Dateinamen in europäische DD-MM-YYYY-Datumsangaben

-- schnipp --

# Stellt Dateinamen mit europäischen Datumsangaben zusammen
euroFilename = beforePart + dayPart + '-' + monthPart + '-' + yearPart +
              afterPart ①
# Ruft die vollständigen absoluten Dateipfade ab
absWorkingDir = os.path.abspath('.')
amerFilename = os.path.join(absWorkingDir, amerFilename)
euroFilename = os.path.join(absWorkingDir, euroFilename)

# Benennt die Dateien um
print('Renaming "%s" to "%s"' % (amerFilename, euroFilename)) ②
#shutil.move(amerFilename, euroFilename) # Nach dem Test entkommentieren
③
```

Speichern Sie den verketteten String in der Variablen euroFilename (①) und übergeben Sie dann den ursprünglichen Dateinamen aus amerFilename und den neuen aus euroFilename an die Funktion shutil.move(), um die Datei umzubenennen (③).

Der Aufruf von shutil.move() ist vorläufig auskommentiert; stattdessen werden die umzubenennenden und die neuen Dateinamen ausgegeben (②). Auf diese Weise können Sie zunächst überprüfen, ob die Umbenennungsaktion auch tatsächlich richtig erfolgt. Nach diesem Test können Sie das Kommentarzeichen vor dem Aufruf von shutil.move() entfernen und das Programm erneut ausführen, um die Dateien tatsächlich umzubenennen.

## Vorschläge für ähnliche Programme

Es gibt noch viele weitere Gründe, um größere Mengen von Dateien umzubennen:

- Um dem Dateinamen ein Präfix voranzustellen, beispielsweise *spam\_*, um aus *eggs.txt* den Namen *spam\_eggs.txt* zu machen
- Um europäische Datumsangaben in Dateinamen in amerikanische umzuwandeln
- Um Nullen aus Dateinamen wie *spam0042.txt* zu entfernen

## Projekt: Einen Ordner in einer ZIP-Datei sichern

Nehmen wir an, Sie haben alle Dateien eines Projekts im Ordner *C:\AlsPythonBook* gespeichert. Da Sie all die Arbeit, die Sie in das Projekt investiert haben, nur ungern verlieren möchten, wollen Sie »Momentaufnahmen« dieses Ordners in Form von ZIP-Dateien anlegen. Da Sie außerdem unterschiedliche Versionen aufbewahren möchten, sollen die Dateinamen dieser ZIP-Dateien eine laufende Nummer aufweisen, also etwa *AlsPythonBook\_1.zip*, *AlsPythonBook\_2.zip* usw. Das könnten Sie zwar auch manuell machen, allerdings ist das eine eher lästige Aufgabe. Außerdem besteht die Gefahr, dass Sie die ZIP-Dateien versehentlich falsch nummerieren. Es wäre viel einfacher, wenn ein Programm diese langweilige Arbeit für Sie übernehmen würde.

Öffnen Sie für dieses Projekt ein neues Dateieditorfenster und speichern Sie das Programm als *backupToZip.py*.

### Schritt 1: Die Namen der ZIP-Dateien bestimmen

In diesem Programm schreiben wir den Code in die Funktion *backupToZip()*, um die Komprimierung auch ganz einfach in anderen Programmen zur Verfügung stellen zu können. Am Ende des Programms wird die Funktion aufgerufen, um die Sicherung vorzunehmen. Schreiben Sie zunächst folgenden Code:

```
#! python3
# backupToZip.py - Kopiert einen Ordner und seinen gesamten Inhalt in eine
# ZIP-Datei mit laufender Nummerierung

import zipfile, os ❶

def backupToZip(folder):
    # Sichert den ganzen Inhalt von "folder" in einer ZIP-Datei

    folder = os.path.abspath(folder) # "folder" muss ein absoluter Pfad sein
```

```
# Ermittelt aufgrund der bereits vorhandenen Dateinamen den Namen für die
# aktuelle Datei
number = 1 ❷
while True: ❸
    zipFilename = os.path.basename(folder) + '_' + str(number) + '.zip'
    if not os.path.exists(zipFilename):
        break
    number = number + 1

# TODO: ZIP -Datei erstellen ❹

# TODO: Den Verzeichnisbaum durchlaufen und alle Dateien in allen Ordnern
# komprimieren
print('Done.')
```

backupToZip('C:\\delicious')

Als Erstes kümmern Sie sich um die Grundlagen, nämlich die Shebang-Zeile (#!), eine Beschreibung, wozu das Programm da ist, und den Import der Module `zipfile` und `os` (❶).

Anschließend definieren Sie die Funktion `backToZip()`. Ihr einziger Parameter ist der String `folder`, der den Pfad zu dem zu komprimierenden Ordner angibt. Die Funktion ermittelt den Dateinamen für die neue ZIP-Datei, erstellt diese, durchläuft den in `folder` angegebenen Ordner und fügt alle darin enthaltenen Unterordner und Dateien zu der ZIP-Datei hinzu. Für die anderen Schritte, die noch zu erledigen sind, geben Sie vorläufig `TODO`-Kommentare als Gedächtnisstütze im Code an (❷).

Für die Benennung der ZIP-Datei greift die Funktion auf den Grundnamen des in `folder` angegebenen absoluten Pfads zurück. Soll beispielsweise der Ordner `C:\\delicious` gesichert werden, lautet der Name der ZIP-Datei `delicious_N.zip`, wobei `N` bei der ersten Ausführung des Programms 1 ist, bei der zweiten Ausführung 2 usw.

Den Wert von `N` ermitteln Sie, indem Sie prüfen, ob `delicious_1.zip` bereits vorhanden ist; wenn ja, müssen Sie nachsehen, ob es auch `delicious_2.zip` schon gibt, usw. Für `N` wird die Variable `number` verwendet (❸). In der Schleife, die `os.path.exists()` aufruft, um zu prüfen, ob die Dateinamen schon vorhanden sind, wird diese Variable ständig inkrementiert (❹). Bei dem ersten noch nicht vorhandenen Dateinamen wird die Schleife mit `break` abgebrochen. Damit haben wir den Dateinamen für das neue ZIP-Archiv gefunden.

## Schritt 2: Die neue ZIP-Datei erstellen

Als Nächstes geht es daran, die ZIP-Datei tatsächlich anzulegen. Das Programm muss jetzt wie folgt aussehen:

```
#! python3
# backupToZip.py - Kopiert einen Ordner und seinen gesamten Inhalt in eine
# ZIP-Datei mit laufender Nummerierung

-- schnipp --
while True:
    zipfilename = os.path.basename(folder) + '_' + str(number) + '.zip'
    if not os.path.exists(zipfilename):
        break
    number = number + 1

# Erstellt die ZIP-Datei.
print('Creating %s...' % (zipfilename))
backupZip = zipfile.ZipFile(zipfilename, 'w') ❶

# TODO: Den Verzeichnisbaum durchlaufen und alle Dateien in allen Ordnern
# komprimieren
print('Done.')

backupToZip('C:\\\\delicious')
```

Da der vorgesehene Name für die neue ZIP-Datei jetzt in der Variablen `zipfilename` gespeichert ist, können Sie `zipfile.ZipFile()` aufrufen, um das Archiv tatsächlich anzulegen (❶). Als zweites Argument müssen Sie dabei '`w`' übergeben, damit die ZIP-Datei im Schreibmodus geöffnet wird.

## Schritt 3: Den Verzeichnisbaum durchlaufen und Inhalte zur ZIP-Datei hinzufügen

Nun müssen wir die Funktion `os.walk()` verwenden, um sämtliche Dateien in dem Ordner und allen seinen Unterordnern zu durchlaufen. Ergänzen Sie das Programm wie folgt:

```
#! python3
# backupToZip.py - Kopiert einen Ordner und seinen gesamten Inhalt in eine
# ZIP-Datei mit laufender Nummerierung

-- schnipp --
```

```
# Durchläuft den Verzeichnisbaum und komprimiert alle Dateien in allen
# Ordnern
for foldername, subfolders, filenames in os.walk(folder):    ❶
    print('Adding files in %s...' % (foldername))
    # Fügt den aktuellen Ordner zur ZIP-Datei hinzu
    backupZip.write(foldername)    ❷
    # Fügt alle Dateien in diesem Ordner zur ZIP-Datei hinzu
    for filename in filenames:    ❸
        newBase = os.path.basename(folder) + '_'
        if filename.startswith(newBase) and filename.endswith('.zip'):
            continue # don't backup the backup ZIP files
        backupZip.write(os.path.join(foldername, filename))
    backupZip.close()
print('Done.')

backupToZip('C:\\delicious')
```

Wenn Sie die Funktion `os.walk()` in einer `for`-Schleife verwenden (❶), gibt sie in jeder Iteration den Namen des aktuellen Ordners sowie die Namen der darin enthaltenen Dateien und Unterordner zurück.

In der `for`-Schleife wird der laufende Ordner zu der ZIP-Datei hinzugefügt (❷). Die verschachtelte `for`-Schleife geht alle Dateinamen in der Liste `filenames` durch (❸). Alle auf diese Weise gefundenen Dateien werden der ZIP-Datei hinzugefügt. Die einzigen Ausnahmen bilden zuvor angelegte ZIP-Archive.

Bei der ersten Ausführung gibt dieses Programm Folgendes aus:

```
Creating delicious_1.zip...
Adding files in C:\\delicious...
Adding files in C:\\delicious\\cats...
Adding files in C:\\delicious\\waffles...
Adding files in C:\\delicious\\walnut...
Adding files in C:\\delicious\\walnut\\waffles...
Done.
```

Bei der zweiten Ausführung werden alle Dateien in `C:\\delicious` in das ZIP-Archiv `delicious_2.zip` gestellt usw.

## Vorschläge für ähnliche Programme

Verzeichnisbäume zu durchlaufen und Dateien zu komprimierten ZIP-Archiven hinzuzufügen, kann auch in anderen Programmen sinnvoll sein, die beispielsweise folgende Aufgaben erfüllen:

- Einen Verzeichnisbaum durchlaufen und nur Dateien mit bestimmten Erweiterungen archivieren, z. B. ausschließlich `.txt`- und `.py`-Dateien, aber keine anderen
- Einen Verzeichnisbaum durchlaufen und alle Dateien außer denen mit bestimmten Erweiterungen archivieren, z. B. alles außer `.txt`- und `.py`-Dateien
- In einem Verzeichnisbaum den Ordner finden, der die meisten Dateien enthält oder den meisten Platz auf der Festplatte einnimmt

## Zusammenfassung

Selbst erfahrene Computerbenutzer nehmen Arbeiten an Dateien häufig manuell mit Maus und Tastatur vor. In den Datei-Explorern der modernen Betriebssysteme ist der Umgang mit Dateien auch sehr einfach, sofern es nur um wenige geht. Allerdings gibt es Aufgaben, die bei manueller Erledigung im Datei-Explorer Stunden in Anspruch nehmen können.

In den Modulen `os` und `shutil` gibt es Funktionen, um Dateien zu kopieren, zu verschieben, umzubenennen und zu löschen, wobei Sie zum Löschen jedoch lieber auf das Modul `send2trash` ausweichen sollten, das die Dateien in den Papierkorb verschiebt, anstatt sie sofort endgültig zu entfernen. Wenn Sie Programme zur Dateiverwaltung schreiben, sollten Sie den Code, der den eigentlichen Kopier-, Verschiebungs-, Umbenennungs- oder Löschvorgang ausführt, vorläufig auskommentieren und eine `print()`-Anweisung hinzufügen, um in einem Testlauf des Programms zunächst einmal zu überprüfen, ob es auch wirklich das macht, was es tun soll.

Oft müssen Sie solche Tätigkeiten nicht nur an einzelnen Dateien oder Ordnern vornehmen, sondern an allen Unterordnern eines Ordners, allen Unterordnern dieser Unterordner usw. Diese Expedition durch die Ordnerstruktur nimmt Ihnen die Funktion `os.walk()` ab. In Ihrem Programm brauchen Sie dann nur noch anzugeben, was mit den Dateien geschehen soll, die bei diesem Durchlauf entdeckt werden.

Das Modul `zipfile` bietet die Möglichkeit, Dateien in Python-Programmen zu ZIP-Archiven zu komprimieren und solche Archive zu entpacken. Zusammen mit den Dateiverwaltungsfunktionen von `os` und `shutil` können Sie damit mehrere Dateien an beliebigen Speicherorten Ihrer Festplatte in ZIP-Dateien zusammen-

fassen. Solche ZIP-Dateien lassen sich viel leichter auf Websites hochladen oder als E-Mail-Anhänge verschicken als einzelne Dateien.

In diesem Buch finden Sie eine Menge Beispielcode, den Sie einfach kopieren können. Wenn Sie aber eigene Programme schreiben, werden Sie wahrscheinlich nicht gleich auf Anhieb alles richtig machen. Im nächsten Kapitel lernen Sie einige Python-Module kennen, mit denen Sie Ihre Programme analysieren und debuggen können, damit sie fehlerfrei laufen.

## **Wiederholungsfragen**

1. Was ist der Unterschied zwischen `shutil.copy()` und `shutil.copytree()`?
2. Mit welcher Funktion benennen Sie Dateien um?
3. Was ist der Unterschied zwischen den Löschfunktionen in den Modulen `send2trash` und `shutil`?
4. Ebenso wie `File`-Objekte verfügen auch `ZipFile`-Objekte über die Methode `close()`. Welche `ZipFile`-Methode entspricht aber der `File`-Methode `open()`?

## **Übungsprojekte**

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erledigen:

### **Selektives Kopieren**

Schreiben Sie ein Programm, das einen Verzeichnisbaum durchläuft und dabei nach Dateien mit einer bestimmten Endung sucht (z. B. `.pdf` oder `.jpg`). Kopieren Sie die Dateien in einen neuen Ordner.

### **Nicht mehr benötigte Dateien löschen**

Es kommt oft vor, dass einige wenige überflüssige, aber riesige Dateien oder Ordner viel Platz auf der Festplatte einnehmen. Wenn Sie Speicherplatz auf dem Rechner freimachen wollen, können Sie besonders viel ausrichten, indem Sie die größten dieser unnötigen Dateien löschen. Dazu müssen Sie sie aber zunächst einmal finden.

Schreiben Sie ein Programm, das einen Verzeichnisbaum durchläuft und dabei nach außergewöhnlich großen Dateien und Ordnern sucht, z. B. solchen mit einer Dateigröße von mehr als 100 MB. (Um die Dateigröße zu ermitteln, können Sie die Funktion `os.path.getsize()` aus dem Modul `os` verwenden.) Geben Sie die Namen und absoluten Pfade dieser Dateien auf dem Bildschirm aus.

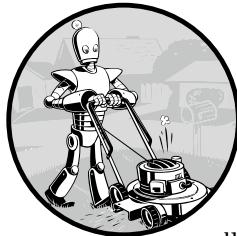
### Lücken entfernen

Schreiben Sie ein Programm, das in einem bestimmten Ordner alle Dateien findet, deren Namen ein bestimmtes Präfix und eine laufende Nummer aufweisen, z. B. *spam001.txt*, *spam002.txt* usw. Wenn es bei der Nummerierung Lücken gibt (wenn es also beispielsweise die Dateien *spam001.txt* und *spam003.txt* gibt, aber nicht *spam002.txt*), soll es die Dateien mit den höheren Nummern umbenennen, um diese Lücken zu schließen.

Als Zusatzaufgabe schreiben Sie ein anderes Programm, das Lücken in die Nummerierung von Dateien einfügt, sodass neue Dateien hinzugefügt werden können.

# 10

## Debugging



Sie wissen nun bereits genug, um auch anspruchsvollere Programme zu schreiben – in denen allerdings auch schwerer zu findende Bugs stecken können. In diesem Kapitel lernen Sie daher einige Werkzeuge und Techniken kennen, um die Ursachen von Fehlern in Ihren Programmen schneller und mit weniger Aufwand zu finden und auszumerzen.

Unter Programmierern kursiert der alte Witz: »Den Code zu schreiben, macht 90 % der Programmierung aus. Für das Debugging gehen die restlichen 90 % drauf.«

Ein Computer macht nur das, was Sie ihm sagen. Er kann Ihre Gedanken nicht lesen und das tun, was Sie *beabsichtigen* wollten. Auch professionellen Programmierern unterlaufen ständig Fehler. Seien Sie daher nicht entmutigt, wenn in Ihrem Programm Probleme auftreten.

Zum Glück gibt es einige Werkzeuge und Techniken, um herauszufinden, was der Code genau macht und wo etwas schiefläuft. Als Erstes sehen wir uns die Protokollierung und Zusicherungen an, mit denen sich Bugs schon frühzeitig erkennen lassen. Je eher Sie einen Fehler aufspüren, umso leichter lässt er sich im Allgemeinen beheben.

Danach werfen wir einen genaueren Blick auf den Debugger. Dabei handelt es sich um eine Einrichtung von IDLE, die ein Programm Anweisung für Anweisung ausführt, sodass Sie die Gelegenheit haben, sich die Werte anzusehen, die die Variablen nacheinander annehmen. Das dauert natürlich viel länger als eine normale Ausführung des Programms, ist aber sehr hilfreich, da Sie dadurch die wirklichen Werte einsehen können und nicht darauf beschränkt sind, aus dem Quellcode zu schließen, wie sie eigentlich aussehen müssten.

## Ausnahmen auslösen

Bei der Ausführung von ungültigem Code löst Python eine Ausnahme aus. In Kapitel 3 haben Sie schon erfahren, wie Sie die Anweisungen `try` und `except` nutzen können, um zu erwartende Ausnahmen vorwegzunehmen und dafür zu sorgen, dass das Programm trotzdem weiterhin ausgeführt wird. Es ist aber auch möglich, im Code eigene Ausnahmen auszulösen. Das bedeutet so viel wie: »Hör auf, den Code in dieser Funktion auszuführen, und fahre mit der `except`-Anweisung fort.«

Um Ausnahmen auszulösen, verwenden Sie eine `raise`-Anweisung. Sie besteht aus folgenden Elementen:

- Dem Schlüsselwort `raise`
- Einem Aufruf der Funktion `Exception()`
- Einem String mit einer Fehlermeldung, der an `Exception()` übergeben wird

Probieren Sie zur Veranschaulichung folgenden Code in der interaktiven Shell aus:

```
>>> raise Exception('This is the error message.')
Traceback (most recent call last):
  File "<pyshell#191>", line 1, in <module>
    raise Exception('This is the error message.')
Exception: This is the error message.
```

Da es hier keine `try`- und `except`-Anweisungen gibt, um die von `raise` ausgelöste Ausnahme abzufangen, stürzt das Programm ab und zeigt die angegebene Fehlermeldung an.

Die Ausnahmebehandlung erfolgt meistens nicht in der betroffenen Funktion selbst, sondern in dem Code, der die Funktion auruft. Daher wird die Anweisung `raise` häufig in einer Funktion verwendet, während die `try`- und `except`-Anweisungen in dem aufrufenden Code stehen. Geben Sie dazu den folgenden Code im Dateieditor ein und speichern Sie ihn als `boxPrint.py`:

```
def boxPrint(symbol, width, height):
    if len(symbol) != 1:
        raise Exception('Symbol must be a single character string.') ❶
```

```
if width <= 2:  
    raise Exception('Width must be greater than 2.') ❷  
if height <= 2:  
    raise Exception('Height must be greater than 2.') ❸  
print(symbol * width)  
for i in range(height - 2):  
    print(symbol + (' ' * (width - 2)) + symbol)  
print(symbol * width)  
  
for sym, w, h in (('*', 4, 4), ('0', 20, 5), ('x', 1, 3), ('ZZ', 3, 3)):  
    try:  
        boxPrint(sym, w, h)  
    except Exception as err: ❹  
        print('An exception happened: ' + str(err)) ❺
```

Hier wird zunächst die Funktion `boxPrint()` definiert, die ein Zeichen, eine Breiten- und eine Höhenangabe entgegennimmt und mit dem angegebenen Symbol das Bild eines Kastens mit den angegebenen Abmessungen erstellt und in der Konsole ausgibt.

Um damit sinnvoll einen Kasten zeichnen zu können, darf nur ein einzelnes Zeichen übergeben werden und sowohl die Breite als auch die Höhe müssen größer als 2 sein. In den `if`-Anweisungen lösen wir jeweils Ausnahmen aus, wenn eine dieser Voraussetzungen nicht erfüllt ist. Im zweiten Teil des Codes wird `boxPrint()` mit verschiedenen Argumenten aufgerufen. Ungültige Argumente werden dabei von dem `try/except`-Block gehandhabt.

In diesem Programm wird die Anweisung `except` in der Form `except Exception as err` verwendet (❹). Wenn `boxPrint()` ein `Exception`-Objekt zurückgibt (❶) (❷) (❸), speichert `except` dieses Objekt in der Variablen `err`. Anschließend wird das Objekt an `str()` übergeben und in einen String umgewandelt, um eine Fehlermeldung für den Benutzer anzuzeigen (❺). Die Ausgabe des Programms sieht wie folgt aus:

```
****  
* *  
* *  
****  
000000000000000000000000  
0          0  
0          0  
0          0  
000000000000000000000000  
An exception happened: Width must be greater than 2.  
An exception happened: Symbol must be a single character string.
```

Mit `try` und `except` können Sie Fehler auf elegante Weise abfangen, sodass das Programm nicht einfach abstürzt.

## Traceback als String abrufen

Wenn Python auf einen Fehler stößt, ruft es eine *Rückverfolgung (Traceback)* hervor, die einen wahren Schatz an Informationen bietet. Sie enthält die Fehlermeldung, die Nummer der Codezeile, die den Fehler verursacht hat, und die Folge der Funktionsaufrufe, die zu dem Fehler führten. Letzteres wird als *Aufrufstack* oder *Callstack* bezeichnet.

Geben Sie das folgende Programm im Dateieditor ein und speichern Sie es als *errorExample.py*:

```
def spam():
    bacon()
def bacon():
    raise Exception('This is the error message.')
spam()
```

Wenn Sie dieses Programm ausführen, erhalten Sie folgende Ausgabe:

```
Traceback (most recent call last):
  File "errorExample.py", line 7, in <module>
    spam()
  File "errorExample.py", line 2, in spam
    bacon()
  File "errorExample.py", line 5, in bacon
    raise Exception('This is the error message.')
Exception: This is the error message.
```

Aus diesem Traceback können Sie ablesen, dass der Fehler in Zeile 5 auftrat, und zwar in der Funktion `bacon()`. Dieser Aufruf von `bacon()` erfolgte in Zeile 2 in der `spam()`, die wiederum in Zeile 7 aufgerufen wird. In Programmen, in denen Funktionen an verschiedenen Stellen aufgerufen werden können, ist der Aufrufstack sehr hilfreich, um herauszufinden, welcher Aufruf zu einem Fehler geführt hat.

Python zeigt einen solchen Traceback immer dann an, wenn eine Ausnahme ausgelöst, aber nicht behandelt wird. Sie können den Traceback-String aber auch mit `traceback.format_exc()` abrufen. Diese Funktion ist praktisch, wenn Sie die Ausnahme mithilfe einer `except`-Anweisung behandeln, aber trotzdem auf die Informationen aus dem Traceback zugreifen möchten. Bevor Sie diese Funktion aufrufen können, müssen Sie das Python-Modul `traceback` importieren.

Anstatt das Programm abstürzen zu lassen, wenn eine Ausnahme auftritt, können Sie die Traceback-Informationen in eine Protokolldatei schreiben und das Programm durch eine Ausnahmebehandlung am Laufen halten. Wenn Sie das Programm später debuggen, können Sie dazu die Informationen aus der Protokolldatei heranziehen. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import traceback
>>> try:
        raise Exception('This is the error message.')
except:
    errorFile = open('errorInfo.txt', 'w')
    errorFile.write(traceback.format_exc())
    errorFile.close()
    print('The traceback info was written to errorInfo.txt.')
116
The traceback info was written to errorInfo.txt.
```

Da 116 Zeichen in die Datei geschrieben wurden, gibt die Methode `write()` den Wert 116 zurück. Der Text des Traceback steht jetzt in `errorInfo.txt`.

```
Traceback (most recent call last):
  File "<pyshell#28>", line 2, in <module>
Exception: This is the error message.
```

## Zusicherungen (Assertions)

Eine *Zusicherung (Assertion)* ist eine Plausibilitätsprüfung, um sicherzustellen, dass der Code nicht irgendetwas offensichtlich Falsches macht. Durchgeführt wird eine solche Prüfung mithilfe der Anweisung `assert`. Schlägt sie fehl, so wird die Ausnahme `AssertionError` ausgelöst. Eine `assert`-Anweisung setzt sich wie folgt zusammen:

- Das Schlüsselwort `assert`
- Eine Bedingung (also ein Ausdruck, der zu `True` oder `False` ausgewertet wird)
- Ein Komma
- Ein String, der angezeigt werden soll, wenn die Bedingung `False` ist

Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> podBayDoorStatus = 'open'
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
>>> podBayDoorStatus = 'I\'m sorry, Dave. I\'m afraid I can\'t do that.'
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
AssertionError: The pod bay doors need to be "open".
```

Zu Anfang haben wir `podBayDoorStatus` auf `'open'` gesetzt, sodass wir anschließend davon ausgehen, dass diese Variable den Wert `'open'` hat. In einem Programm, das diese Variable nutzt, kann es viel Code geben, für den dieser Wert tatsächlich `'open'`

sein muss, damit er wie erwartet funktioniert. Daher fügen wir eine Zusicherung hinzu, um dafür zu sorgen, dass diese Voraussetzung gegeben ist. Außerdem fügen wir die Meldung 'The pod bay doors need to be "open".' hinzu, damit wir auf einen Blick erkennen können, was schiefgelaufen ist, wenn die Zusicherung nicht erfüllt sein sollte.

Es kann sein, dass wir `podBayDoorStatus` im weiteren Verlauf einen anderen Wert zuweisen. Unter der Menge an Codezeilen eines umfangreichen Programms mag dieser Fehler trotz seiner Offensichtlichkeit nicht auffallen. Die Zusicherung aber erkennt diesen Fehler und sagt uns, was hier nicht stimmt.

Im Grunde genommen sagt eine `assert`-Anweisung Folgendes aus: »Ich versichere, dass diese Bedingung wahr ist; wenn nicht, dann steckt irgendwo in dem Programm ein Fehler.« Im Gegensatz zu Ausnahmen darf der Code fehlgeschlagene `assert`-Anweisungen nicht mit `try` und `except` abzufangen versuchen, denn wenn eine Zusicherung nicht erfüllt ist, dann *soll* das Programm abstürzen. Durch diesen zwangsweisen Absturz verkürzen Sie die Zeit zwischen dem Auftreten der Ursache und dem Punkt, an dem der Fehler offensichtlich wird. Dadurch müssen Sie weniger Code überprüfen, um die Stelle mit dem Bug zu finden.

Zusicherungen sind für Programmierfehler da, nicht für Benutzerfehler. Fehler, von denen sich das Programm erholen kann (beispielsweise eine Datei, die nicht vorhanden ist, oder ungültige Benutzereingaben), spüren Sie nicht mit `assert` auf, sondern Sie fangen sie mit einer Ausnahme ab.

## Zusicherungen in einem Ampelsimulator

Stellen Sie sich ein Programm zur Simulation von Ampelschaltungen vor. Die Datenstruktur zur Darstellung der Ampeln an einer Straßenkreuzung ist ein Dictionary mit den Schlüsseln '`ns`' und '`ew`' für die Fahrtrichtungen Nord-Süd bzw. Ost-West. Zulässige Werte für diese Schlüssel sind '`green`', '`yellow`' und '`red`'. Diese Datenstruktur kann beispielsweise wie folgt aussehen:

```
market_2nd = {'ns': 'green', 'ew': 'red'}
mission_16th = {'ns': 'red', 'ew': 'green'}
```

Diese beiden Variablen stehen für die Kreuzungen Market Street/2nd Street und Mission Street/16th Street. Für das Simulationsprogramm müssen Sie die Funktion `switchLights()` schreiben, die das Dictionary für eine Kreuzung entgegennimmt und die Ampeln schaltet.

Als Erstes mag sich die Idee aufdrängen, dass die Funktion `switchLights()` die Ampeln einfach zur nächsten Farbe in der Reihenfolge weiterschalten könnte, indem sie alle '`green`'-Werte in '`yellow`' ändert, alle '`yellow`'-Werte in '`red`' und alle '`red`'-Werte in '`green`'. Der Code dafür sähe wie folgt aus:

```
def switchLights(stoplight):
    for key in stoplight.keys():
        if stoplight[key] == 'green':
            stoplight[key] = 'yellow'
        elif stoplight[key] == 'yellow':
            stoplight[key] = 'red'
        elif stoplight[key] == 'red':
            stoplight[key] = 'green'

switchLights(market_2nd)
```

Vielleicht fällt Ihnen jetzt schon auf, was bei diesem Code nicht stimmt, aber nehmen wir an, Sie hätten schon den ganzen Rest des Simulationscodes geschrieben, der Tausende von Zeilen umfasst, ohne es zu bemerken. Wenn Sie das Programm ausführen, stürzt es zwar nicht ab, verursacht aber eine Massenkarambolage der virtuellen Autos.

Da schon der ganze Rest des Programms vorhanden ist, haben Sie keine Ahnung, wo der Fehler stecken könnte. Vielleicht befindet er sich in dem Simulationscode für die Autos oder für die Fahrer. Es könnte Stunden dauern, um das Problem auf die Funktion `switchLights()` zurückzuführen.

Sie können aber schon beim Schreiben von `switchLight()` die Zusicherung hinzufügen, dass *immer mindestens eine Ampel rot zeigen muss*. Das können Sie mit folgender Zeile am Ende der Funktion erreichen:

```
assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)
```

Wenn diese Zusicherung vorhanden ist, stürzt das Programm mit folgender Fehlermeldung ab:

```
Traceback (most recent call last):
  File "carSim.py", line 14, in <module>
    switchLights(market_2nd)
  File "carSim.py", line 13, in switchLights
    assert 'red' in stoplight.values(), 'Neither light is red! ' +
          str(stoplight)
AssertionError: Neither light is red! {'ns': 'yellow', 'ew': 'green'} ❶
```

Die wichtige Zeile ist hier die mit dem `AssertionError` (❶). Es ist zwar kein schönes Erlebnis, wenn das Programm abstürzt, aber dadurch werden Sie sofort darauf aufmerksam gemacht, dass eine Plausibilitätsprüfung fehlgeschlagen ist: Keine Fahrtrichtung hat Rot, weshalb Autos von allen Seiten in die Kreuzung einfahren können. Durch diesen Test in einem frühen Stadium der Programmausführung können Sie sich eine Menge Arbeit beim Debugging sparen.

## Zusicherungen deaktivieren

Zusicherungen lassen sich deaktivieren, indem Sie bei der Ausführung von Python die Option `-O` übergeben. Das ist praktisch, wenn Sie ein Programm fertiggeschrieben und getestet haben und es nicht durch Plausibilitätsprüfungen ausbremsen lassen wollen (wobei `assert`-Anweisungen jedoch meistens keine merklichen Verzögerungen hervorrufen). Zusicherungen sind für die Entwicklung da, nicht für das fertige Produkt. Wenn Sie das Programm für die Benutzer zur Verfügung stellen, sollte es frei von Bugs sein und keine Plausibilitätsprüfungen mehr benötigen. Einzelheiten darüber, wie Sie ein von Plausibilitätsfehlern höchstwahrscheinlich freies Programm mit `-O` starten, erfahren Sie in Anhang B.

## Protokollierung

Wenn Sie in Ihrem Code eine `print()`-Anweisung verwenden, um während der Ausführung des Programms den Wert einer Variablen auszugeben, führen Sie eine Form von *Protokollierung* für das Debugging durch. Die Protokollierung ist ein hervorragendes Mittel, um herauszufinden, was in einem Programm geschieht und in welcher Reihenfolge es passiert. Mit dem Python-Modul `logging` können Sie eigene Meldungen formulieren und deren Verlauf aufzeichnen. Diese Meldungen geben an, wann die Programmausführung den Aufruf der Protokollierungsfunktion erreicht hat und welche Variablen zu diesem Zeitpunkt spezifiziert waren. Wird eine vorbereitete Protokollmeldung nicht angezeigt, können Sie daraus schließen, dass der betreffende Teil des Codes gar nicht ausgeführt, sondern übersprungen wurde.

### Das Modul `logging` verwenden

Um das Modul `logging` zu verwenden, mit dem Sie während der Ausführung des Programms Protokollmeldungen auf dem Bildschirm ausgeben können, fügen Sie folgenden Code ganz vorn in Ihrem Programm ein (aber hinter der Zeile `#!/python`):

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s
- %(message)s')
```

Es ist nicht unbedingt nötig, genau zu verstehen, was hier im Einzelnen geschieht. Allgemein gesagt: Wenn Python ein Ereignis protokolliert, erstellt es ein `LogRecord`-Objekt mit Informationen über das Ereignis. Die Funktion `basicConfig()` des Moduls `logging` ermöglicht es Ihnen, anzugeben, welche Einzelheiten über das `LogRecord`-Objekt Sie einsehen wollen und wie sie angezeigt werden sollen.

Nehmen wir an, Sie haben eine Funktion geschrieben, um die *Fakultät* einer Zahl zu berechnen. Beispielsweise berechnen Sie die Fakultät von 4 als  $1 \times 2 \times 3 \times 4 = 24$  und die Fakultät von 7 als  $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 = 5040$ . Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *factorialLog.py*. Dieses Programm enthält einen Bug, aber mithilfe der Protokollmeldungen können Sie selbst herausfinden, wo der Fehler liegt.

```
import logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s
- %(message)s')
logging.debug('Start of program')

def factorial(n):
    logging.debug('Start of factorial(%s%%)' % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i is ' + str(i) + ', total is ' + str(total))
    logging.debug('End of factorial(%s%%)' % (n))
    return total

print(factorial(5))
logging.debug('End of program')
```

Immer, wenn wir Protokollinformationen ausgeben wollen, setzen wir die Funktion `logging.debug()` ein. Sie ruft `basicConfig()` auf und gibt die an `debug()` übergebene Meldung sowie eine Zeile mit Informationen in dem Format aus, das wir in `basicConfig()` angegeben haben. Der Aufruf `print(factorial(5))` gehört mit zum eigentlichen Programm, sodass das Ergebnis auch dann ausgegeben wird, wenn die Anzeige von Protokollmeldungen ausgeschaltet ist.

Die Ausgabe sieht wie folgt aus:

```
2015-05-23 16:20:12,664 - DEBUG - Start of program
2015-05-23 16:20:12,664 - DEBUG - Start of factorial(5)
2015-05-23 16:20:12,665 - DEBUG - i is 0, total is 0
2015-05-23 16:20:12,668 - DEBUG - i is 1, total is 0
2015-05-23 16:20:12,670 - DEBUG - i is 2, total is 0
2015-05-23 16:20:12,673 - DEBUG - i is 3, total is 0
2015-05-23 16:20:12,675 - DEBUG - i is 4, total is 0
2015-05-23 16:20:12,678 - DEBUG - i is 5, total is 0
2015-05-23 16:20:12,680 - DEBUG - End of factorial(5)
0
2015-05-23 16:20:12,684 - DEBUG - End of program
```

Die Funktion `factorial()` gibt 0 als Fakultät von 5 zurück, was offensichtlich falsch ist. Die `for`-Schleife soll den Wert in `total` mit den Zahlen von 1 bis 5 multiplizieren.

Die von `logging.debug()` ausgegebenen Protokollmeldungen zeigen aber, dass die Variable `i` bei 0 und nicht bei 1 beginnt. Da ein Produkt von 0 stets 0 ist, enthält `total` auch in allen folgenden Iterationen immer den falschen Wert. Protokollmeldungen zeigen eine Spur auf, der Sie folgen können, um herauszufinden, wo sich ein Fehler eingeschlichen hat.

Ändern Sie die Zeile `for i in range(n + 1):` in `for i in range(1, n + 1):` und führen Sie das Programm erneut aus. Jetzt ergibt sich Folgendes:

```
2015-05-23 17:13:40,650 - DEBUG - Start of program
2015-05-23 17:13:40,651 - DEBUG - Start of factorial(5)
2015-05-23 17:13:40,651 - DEBUG - i is 1, total is 1
2015-05-23 17:13:40,654 - DEBUG - i is 2, total is 2
2015-05-23 17:13:40,656 - DEBUG - i is 3, total is 6
2015-05-23 17:13:40,659 - DEBUG - i is 4, total is 24
2015-05-23 17:13:40,661 - DEBUG - i is 5, total is 120
2015-05-23 17:13:40,661 - DEBUG - End of factorial(5)
120
2015-05-23 17:13:40,666 - DEBUG - End of program
```

Jetzt gibt der Aufruf von `factorial(5)` das korrekte Ergebnis 120 zurück. Anhand der Protokollmeldungen konnten Sie sehen, was innerhalb der Schleife geschah, und den Fehler daher sofort finden.

Wie Sie sehen, gibt `logging.debug()` nicht nur die übergebenen Strings aus, sondern auch einen Zeitstempel und das Wort `DEBUG`.

### Kein Debugging mit `print()`

Da es etwas mühselig ist, import `logging` und `logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')` einzugeben, ist die Versuchung groß, einfach `print()` zu verwenden. Geben Sie dieser Versuchung aber nicht nach! Wenn Sie mit dem Debugging fertig sind, müssen Sie sonst eine Menge Zeit darauf verwenden, die `print()`-Aufrufe für jede einzelne dieser Meldungen wieder aus dem Code zu entfernen. Dabei besteht überdies die Gefahr, dass Sie versehentlich `print()`-Aufrufe für andere Zwecke entfernen. Das Schöne an den Protokollmeldungen des `logging`-Moduls ist, dass Sie in Ihre Programme so viele davon aufnehmen können wie möglich und Sie anschließend alle deaktivieren können, indem Sie ein einziges Mal `logging.disable(logging.CRITICAL)` aufrufen. Anders als bei `print()` können Sie bei der Verwendung von `logging` ganz leicht zwischen Anzeigen und Ausblenden von Protokollmeldungen umschalten.

Protokollmeldungen sind für die Programmierer gedacht, nicht für die Benutzer. Für Sie als Programmierer kann es wichtig sein, den Inhalt eines Dictionary-

Werts zu sehen, um einen Fehler zu finden; für den Benutzer sind solche Informationen unerheblich. Daher verwenden Sie dafür eine Protokollmeldung. Für Meldungen, die der Benutzer sehen soll, etwa *Datei nicht gefunden* oder *Ungültige Eingabe; bitte geben Sie eine Zahl ein!*, sollten Sie dagegen `print()` verwenden. Wenn Sie dann die Protokollmeldungen abschalten, sehen die Benutzer trotzdem noch die Informationen, die sie haben müssen.

## Protokolliergrade

*Protokolliergrade* geben die Wichtigkeit einer Protokollmeldung an. Es gibt fünf dieser Grade. In Tabelle 10–1 sind sie nach steigender Wichtigkeit aufgeführt. Es gibt verschiedene Protokollierfunktionen, um jeweils die Nachrichten eines bestimmten Protokolliergrads zu erfassen.

Protokolliergrad	Protokollierfunktion	Beschreibung
DEBUG	<code>logging.debug()</code>	Der niedrigste Grad. Wird für kleine Einzelheiten verwendet. Diese Meldungen sind gewöhnlich nur dann beachtenswert, wenn Sie Probleme diagnostizieren.
INFO	<code>logging.info()</code>	Dient zur Aufzeichnung von Informationen über allgemeine Ereignisse im Programm oder zur Bestätigung, dass etwas an dem betreffenden Punkt im Programm korrekt funktioniert.
WARNING	<code>logging.warning()</code>	Zeigt mögliche Probleme an, die das Funktionieren des Programms zwar noch nicht beeinträchtigen, es aber in Zukunft tun könnten.
ERROR	<code>logging.error()</code>	Dient zur Aufzeichnung eines Fehlers, durch den das Programm eine bestimmte Aufgabe nicht erledigen konnte.
CRITICAL	<code>logging.critical()</code>	Der höchste Grad. Wird zur Anzeige von fatalen Fehlern verwendet, die die Programmausführung anhalten oder unmittelbar zum Anhalten bringen.

**Tab. 10–1** Protokolliergrade in Python

Ihre Protokollmeldungen übergeben Sie diesen Funktionen als Strings. Die Protokolliergrade stellen dabei nur Vorschläge dar; letzten Endes ist es Ihre Entscheidung, in welche Kategorie Sie Ihre Meldung aufnehmen wollen. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s - %(message)s')
>>> logging.debug('Some debugging details.')
2015-05-18 19:04:26,901 - DEBUG - Some debugging details.
>>> logging.info('The logging module is working.')
2015-05-18 19:04:35,569 - INFO - The logging module is working.
>>> logging.warning('An error message is about to be logged.')
2015-05-18 19:04:56,843 - WARNING - An error message is about to be logged.
>>> logging.error('An error has occurred.')
2015-05-18 19:05:07,737 - ERROR - An error has occurred.
>>> logging.critical('The program is unable to recover!')
2015-05-18 19:05:45,794 - CRITICAL - The program is unable to recover!
```

Mithilfe der Protokolliergrade können Sie angeben, welche Arten von Protokollmeldungen angezeigt werden sollen. Wenn Sie dem Schlüsselwortparameter `level` von `basicConfig()` das Argument `logging.DEBUG` übergeben, werden Meldungen aller Protokolliergrade angezeigt (da `DEBUG` der niedrigste Grad ist). Wenn Sie das Programm schon etwas weiter verbessert haben, sind Sie wahrscheinlich nur noch an Fehlern interessiert. In diesem Fall setzen Sie das `level`-Argument von `basicConfig()` auf `logging.ERROR`. Dadurch werden nur noch `ERROR`- und `CRITICAL`-Meldungen angezeigt, aber keine `DEBUG`-, `INFO`- und `WARNING`-Meldungen mehr.

## Die Protokollierung deaktivieren

Nachdem Sie das Debugging für ein Programm abgeschlossen haben, wollen Sie sicherlich nicht mehr, dass die Protokollmeldungen den Bildschirm überschwemmen. Mit der Funktion `logging.disable()` können Sie sie deaktivieren. Sie müssen also nicht alle Aufrufe der Protokollierfunktionen manuell aus dem Code entfernen, sondern übergeben `logging.disable()` einfach einen Protokolliergrad, woraufhin alle Protokollmeldungen dieses und aller darunter liegenden Grade unterdrückt werden. Um die Protokollierung komplett abzuschalten, fügen Sie Ihrem Programm also einfach die Zeile `logging.disable(logging.CRITICAL)` hinzu:

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO, format=' %(asctime)s -
%(levelname)s - %(message)s')
>>> logging.critical('Critical error! Critical error!')
2015-05-22 11:10:48,054 - CRITICAL - Critical error! Critical error!
>>> logging.disable(logging.CRITICAL)
>>> logging.critical('Critical error! Critical error!')
>>> logging.error('Error! Error!')
```

Da nach `logging.disable()` alle Protokollmeldungen unterdrückt werden, sollten Sie diese Zeile möglichst in die Nähe der Codezeile `import logging` stellen. Da-

durch können Sie den Aufruf auch schnell finden und auskommentieren oder entkommentieren, wenn Sie die Protokollierung schnell ein- bzw. ausschalten wollen.

### Protokollierung in eine Datei

Anstatt die Protokollmeldungen auf dem Bildschirm auszugeben, können Sie sie auch in eine Textdatei schreiben lassen. Dazu nimmt die Funktion `logging.basicConfig()` auch ein Argument mit dem Schlüsselwort `filename` entgegen:

```
import logging
logging.basicConfig(filename='myProgramLog.txt', level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %(message)s')
```

Die Protokollmeldungen werden nun in `myProgramLog.txt` gespeichert. Protokollmeldungen sind zwar sehr hilfreich, können den Bildschirm aber überschwemmen, sodass die Programmausgabe schwer zu lesen ist. Wenn Sie sie stattdessen in eine Datei schreiben lassen, bleibt die Bildschirmausgabe übersichtlich. Außerdem werden die Meldungen gespeichert, sodass Sie sie auch nach der Ausführung des Programms noch lesen können. Die Textdatei können Sie in einem beliebigen Texteditor lesen, z. B. im Windows-Editor oder in TextEdit.

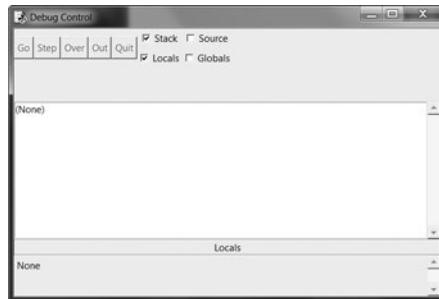
## Der Debugger von IDLE

Der *Debugger* ist eine Einrichtung der IDLE, der ein Programm Zeile für Zeile ausführt und dabei nach jeder Zeile wartet, dass Sie ihn auffordern, weiterzumachen. Dabei können Sie an jedem Punkt im Programmablauf die Werte der Variablen genau unter die Lupe nehmen. Dies ist ein sehr wertvolles Instrument, um Bugs aufzuspüren.

Um den IDLE-Debugger aufzurufen, klicken Sie im Fenster der interaktiven Shell auf *Debug > Debugger*. Dadurch wird das Fenster *Debug Control* eingeblendet, das Sie in Abb. 10–1 sehen.

Aktivieren Sie in diesem Fenster die vier Kontrollkästchen *Stack*, *Locals*, *Source* und *Globals*, damit sämtliche Debuginformationen angezeigt werden. Wenn Sie ein Programm im Dateieditor ausführen und das Fenster *Debug Control* dabei geöffnet ist, hält der Debugger die Ausführung schon vor der ersten Anweisung an und gibt Folgendes aus:

- Die Codezeile, die als Nächstes ausgeführt wird
- Eine Liste aller globalen Variablen und ihrer Werte
- Eine Liste aller lokalen Variablen und ihrer Werte



**Abb. 10-1** Das Fenster *Debug Control*

In der Liste der globalen Variablen werden eine Reihe Variablen angezeigt, die Sie gar nicht definiert haben, darunter `_builtins_`, `_doc_`, `_file_` usw. Diese Variablen legt Python automatisch bei der Ausführung eines Programms fest. Ihre Bedeutung zu erklären, würde den Rahmen dieses Buchs sprengen, aber Sie können sie getrost ignorieren.

Die Programmausführung bleibt so lange angehalten, bis Sie auf eine der fünf Schaltflächen oben im Fenster *Debug Control* klicken: *Go*, *Step*, *Over*, *Out* oder *Quit*.

### Go

Ein Klick auf *Go* führt dazu, dass das Programm normal ausgeführt wird, bis es am Ende angekommen ist oder einen *Haltepunkt* erreicht hat. (Was ein Haltepunkt ist, erfahren Sie weiter hinten in diesem Kapitel.) Wenn Sie das vorgesehene Debugging erledigt haben und das Programm normal fortsetzen wollen, klicken Sie auf *Go*.

### Step

Ein Klick auf *Step* veranlasst den Debugger, die nächste Codezeile auszuführen und dann erneut anzuhalten. Wenn sich dabei die Werte von globalen oder lokalen Variablen ändern, wird die Variablenliste im Fenster *Debug Control* auf den neuen Stand gebracht. Handelt es sich bei der Codezeile um einen Funktionsaufruf, so »steigt« der Debugger in diese Funktion »ein«, d. h., er springt zu deren erster Codezeile.

### Over

Ähnlich wie bei *Step* wird auch bei *Over* die nächste Codezeile ausgeführt. Handelt es sich dabei aber um einen Funktionsaufruf, so »überschreitet« der Debugger den Code in der Funktion, d. h., er führt diesen Code in normaler Geschwindigkeit

aus und hält erst an, nachdem die Funktion die Steuerung wieder zurückgegeben hat. Beispielsweise sind bei einem Aufruf von `print()` die Einzelheiten des Codes in der Python-Funktion `print()` für das Debugging völlig belanglos; Sie wollen nur, dass der übergebene String auf dem Bildschirm ausgegeben wird. Aus diesem Grund verwenden Sie in solchen Fällen *Over* statt *Step*.

## Out

Wenn Sie mit *Step* in eine aufgerufene Funktion eingestiegen sind und nun einfach die Funktion komplett ausführen wollen, können Sie auf *Out* klicken, um aus der Funktion »auszusteigen«. Der Debugger führt dann die folgenden Codezeilen in normaler Geschwindigkeit aus, bis die Funktion die Steuerung wieder zurückgibt.

## Quit

Wenn Sie den Debuggingvorgang abbrechen und den Rest des Programms nicht mehr ausführen wollen, klicken Sie auf *Quit*. Dadurch wird das Programm sofort beendet. Wollen Sie das Programm anschließend normal ausführen, wählen Sie *Debug > Debugger*, um den Debugger auszuschalten.

## Debugging eines Additionsprogramms

Geben Sie im Dateieditor folgenden Code ein:

```
print('Enter the first number to add:')
first = input()
print('Enter the second number to add:')
second = input()
print('Enter the third number to add:')
third = input()
print('The sum is ' + first + second + third)
```

Speichern Sie dieses Programm als *buggyAddingProgram.py* und führen Sie es zunächst ohne Debugger aus. Dabei erhalten Sie folgende Ausgabe:

```
Enter the first number to add:
5
Enter the second number to add:
3
Enter the third number to add:
42
The sum is 5342
```

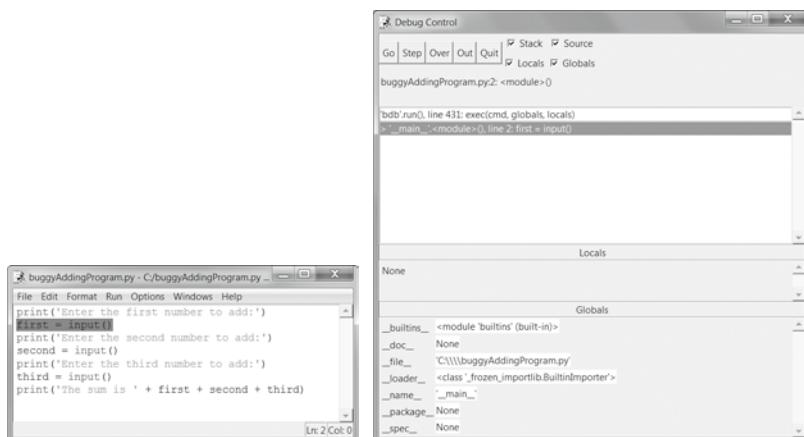
Das Programm ist nicht abgestürzt, aber das Ergebnis ist offensichtlich falsch. Daher rufen wir das Fenster *Debug Control* auf und führen das Programm im Debugger aus.

Wenn Sie **[F5]** drücken oder auf *Run > Run Module* klicken (während *Debug > Debugger* eingeschaltet ist und alle vier Kontrollkästchen im Fenster *Debug Control* aktiviert sind), beginnt die Ausführung im angehaltenen Zustand in Zeile 1. Der Debugger hält immer bei der gerade auszuführenden Zeile an. Das Fenster *Debug Control* sieht jetzt aus wie in Abb. 10–2.



**Abb. 10–2** Das Fenster *Debug Control* beim Start eines Programms mit eingeschaltetem Debugger

Klicken Sie auf *Over*, um den ersten Aufruf von `print()` auszuführen. Hier verwenden Sie *Over* statt *Step*, da Sie schließlich nicht in den Code der Funktion



**Abb. 10–3** Der Dateieditor und das Fenster *Debug Control* nach einem Klick auf *Over*

`print()` einsteigen wollen. Das Fenster *Debug Control* wird auf den Zustand in Zeile 2 aktualisiert und im Dateieditorfenster wird Zeile 2 hervorgehoben (siehe Abb. 10–3), um zu zeigen, wie weit die Programmausführung gekommen ist.

Klicken Sie erneut auf *Over*, um den Aufruf der Funktion `input()` auszuführen. Während IDLE darauf wartet, dass Sie etwas in die interaktive Shell eingeben, stehen die Schaltflächen im Fenster *Debug Control* nicht zur Verfügung. Geben Sie 5 ein und drücken Sie die Eingabetaste. Jetzt können Sie die Schaltflächen in *Debug Control* wieder verwenden.

Klicken Sie weiterhin auf *Over* und geben Sie 3 und 42 als die nächsten Zahlen ein, bis der Debugger Zeile 7 mit dem letzten `print()`-Aufruf des Programms erreicht. Das Fenster *Debug Control* sieht jetzt so aus wie in Abb. 10–4. Im Bereich *Globals* können Sie erkennen, dass die ersten drei Variablen auf die Strings '5', '3' und '42' gesetzt sind. Bei der Ausführung der letzten Zeile werden diese Strings verkettet, anstatt die Zahlenwerte zu addieren. Das ist der Grund für den Fehler.



**Abb. 10–4** Das Fenster *Debug Control* bei der Überprüfung der letzten Zeile. Die Ursache des Fehlers besteht darin, dass die Variablen Strings enthalten.

Das Programm mit dem Debugger schrittweise durchzugehen, ist sehr hilfreich, nimmt aber auch viel Zeit in Anspruch. Oft wollen Sie das Programm einfach ganz normal durchlaufen lassen, bis es einen bestimmten Punkt erreichen. Das können Sie tun, indem Sie im Debugger Haltepunkte einrichten.

## Haltepunkte

*Haltepunkte (Breakpoint)* können Sie in beliebigen Zeilen definieren. Wenn der Debugger bei der Programmausführung die betreffende Zeile erreicht, hält er an. Geben Sie im Dateieditor das folgende Programm ein, das 1000 Münzwürfe simuliert, und speichern Sie es als *coinFlip.py*.

```
import random
heads = 0
for i in range(1, 1001):
    if random.randint(0, 1) == 1:    ❶
        heads = heads + 1
    if i == 500:
        print('Halfway done!')    ❷
print('Heads came up ' + str(heads) + ' times.')
```

Der Aufruf von `random.randint(0, 1)` (❶) gibt in 50 % der Fälle 0 zurück und in den anderen 50 % 1. Damit können wir einen 50:50-Münzwurf simulieren, wobei 1 für »Kopf« steht. Wenn Sie dieses Programm ausführen, erhalten Sie sehr schnell folgende Ausgabe:

```
Halfway done!
Heads came up 490 times.
```

Im Debugger müssten Sie tausend Mal auf *Over* klicken, bevor das Programm endet. Falls Sie nur an dem Wert interessiert sind, den `heads` nach 500 von 1000 Münzwürfen hat, können Sie jedoch einen Haltepunkt in die Zeile `print('Halfway done!')` setzen (❷). Um das zu tun, rechtsklicken Sie im Dateieditor auf die betreffende Zeile und wählen *Set Breakpoint* (siehe Abb. 10–5).

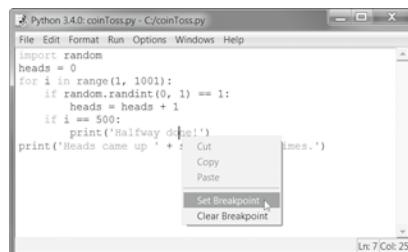


Abb. 10–5 Einen Haltepunkt festlegen

Setzen Sie den Haltepunkt nicht in die Zeile mit der `if`-Anweisung, da sie bei jedem Schleifendurchlauf ausgeführt wird. Da der Haltepunkt aber in dem Code in der `if`-Anweisung steckt, hält der Debugger nur an, wenn die Ausführung die Klausel erreicht.

Im Dateieditor wird die Zeile mit dem Haltepunkt gelb hinterlegt. Wenn Sie das Programm mit eingeschaltetem Debugger starten, hält die Ausführung wie üblich in der ersten Zeile an. Bei einem Klick auf `Go` aber läuft das Programm normal weiter, bis es die Zeile mit dem Haltepunkt erreicht. Danach können Sie wie üblich mit den Schaltflächen `Go`, `Over`, `Step` und `Out` weitermachen.

Um einen Haltepunkt wieder zu entfernen, rechtsklicken Sie im Dateieditor auf die Zeile und wählen *Clear Breakpoint* aus dem Kontextmenü. Die gelbe Hinterlegung verschwindet und der Debugger hält nicht mehr bei dieser Zeile an.

## Zusammenfassung

Zusicherungen, Ausnahmen, Protokollierung und der Debugger sind wertvolle Instrumente, um Bugs in Ihren Programmen aufzuspüren oder zu verhindern. Die Zusicherungen, die Sie mithilfe der Python-Anweisung `assert` einfügen, bieten eine gute Möglichkeit, um Plausibilitätsprüfungen vorzunehmen, durch die Sie frühzeitig gewarnt werden, wenn eine notwendige Bedingung nicht erfüllt ist. Allerdings sind Zusicherungen nur für Fehler da, von denen sich das Programm nicht erholen, sondern bei denen es möglichst rasch abstürzen soll. Für andere Fälle setzen Sie Ausnahmen ein.

Ausnahmen können mit `try`- und `except`-Anweisungen abgefangen werden. Darüber hinaus bietet das Modul `logging` eine gute Möglichkeit, um den Code zu beobachten, während er ausgeführt wird. Da es unterschiedliche Protokolliergrade gibt und das Protokoll in eine Textdatei geschrieben werden kann, ist diese Vorgehensweise besser geeignet als die Verwendung der Funktion `print()`.

Mit dem Debugger können Sie ein Programm Zeile für Zeile durchgehen oder aber die Ausführung nur bei Zeilen anhalten lassen, in die Sie einen Haltepunkt gesetzt haben. Im Debugger können Sie den Zustand aller Variablenwerte zu jedem Zeitpunkt im Ablauf des Programms einsehen.

Diese Werkzeuge und Techniken für das Debugging ermöglichen es Ihnen, Programme zu schreiben, die auch wirklich funktionieren. Beim Schreiben von Code schleichen sich immer Fehler ein. Das ist eine unabänderliche Tatsache, wie viele Jahre an Programmiererfahrung Sie auch immer haben mögen.

## Wiederholungsfragen

1. Schreiben Sie eine assert-Anweisung, die einen AssertionError auslöst, wenn die Integervariable spam kleiner als 10 ist.
2. Schreiben Sie eine assert-Anweisung, die einen AssertionError auslöst, wenn die Variablen eggs und bacon die gleichen Strings enthalten, wobei Unterschiede zwischen Groß- und Kleinschreibung nicht berücksichtigt werden (d. h., dass 'goodbye' und 'GOODbye' als gleich angesehen werden).
3. Schreiben Sie eine assert-Anweisung, die *immer* einen AssertionError auslöst.
4. Welche beiden Zeilen müssen in einem Programm enthalten sein, damit `logging.debug()` aufgerufen werden kann?
5. Welche beiden Zeilen müssen in einem Programm enthalten sein, damit `logging.debug()` eine Protokollmeldung an die Datei *programLog.txt* senden kann?
6. Welche fünf Protokolliergrade gibt es?
7. Mit welcher Codezeile können Sie sämtliche Protokollmeldungen in einem Programm ausschalten?
8. Warum ist es besser, Protokollmeldungen zu verwenden, anstatt die gleichen Meldungen mithilfe der Funktion `print()` auszugeben?
9. Welche Unterschiede gibt es zwischen den Schaltflächen *Step*, *Over* und *Out* im Fenster *Debug Control*?
10. Wann hält der Debugger an, wenn Sie im Fenster *Debug Control* auf *Go* geklickt haben?
11. Was ist ein Haltepunkt?
12. Wie setzen Sie in IDLE einen Haltepunkt in eine Codezeile?

## Übungsprojekt

Debuggen Sie zur Übung das folgende Programm:

### Münzwurfprogramm

Das folgende Programm soll ein einfaches Münzwurfspiel simulieren. Der Spieler hat zwei Versuche (es ist wirklich ein einfaches Spiel). Allerdings enthält dieses Programm einige Bugs. Führen Sie es einige Male aus, um die Fehler zu finden, die die korrekte Ausführung verhindern.

```
import random
guess = ''
while guess not in ('heads', 'tails'):
    print('Guess the coin toss! Enter heads or tails:')
    guess = input()
toss = random.randint(0, 1) # 0 is tails, 1 is heads
if toss == guess:
    print('You got it!')
else:
    print('Nope! Guess again!')
guesss = input()
if toss == guess:
    print('You got it!')
else:
    print('Nope. You are really bad at this game.') 
```



# 11

## Web Scraping



In den seltenen, erschreckenden Augenblicken, in denen ich keine drahtlose Internetverbindung habe, wird mir bewusst, dass ich einen Großteil meiner Arbeit am Computer in Wirklichkeit im Internet erledige. Es ist für mich schon zur Gewohnheit geworden, meinen E-Mail-Posteingang zu überprüfen, die Tweets meiner Freunde zu lesen oder nach Antworten auf Fragen wie »Hatte Kurtwood Smith vor dem ursprünglichen Robocop-Film von 1987 eigentlich irgendwelche anderen größeren Rollen?«<sup>1</sup> zu forschen.

Es wäre daher schön, wenn auch unsere Programme online gehen könnten. Der Begriff *Web Scraping* (»Schürfen im Web«) wird für Programme verwendet, die Inhalte aus dem Web herunterladen und verarbeiten. Google unterhält eine Menge solcher Programme, um Webseiten für seine Suchmaschine zu indizieren. In diesem Kapitel lernen Sie verschiedene Module kennen, mit denen Web Scraping in Python möglich wird:

<sup>1</sup> Die Antwort ist nein.

- `webbrowser` Dieses Modul ist im Lieferumfang von Python enthalten und öffnet eine bestimmte Seite im Browser.
- `requests` Lädt Dateien und Webseiten aus dem Internet herunter.
- `Beautiful Soup` Analysiert oder *parst* HTML, das Format, in dem die meisten Webseiten geschrieben sind.
- `Selenium` Startet und steuert einen Webbrower. Damit können Sie in dem Browser auch Formulare ausfüllen und Mausklicks simulieren.

## Projekt: `mapIt.py` mit dem Modul `webbrowser`

Die Funktion `open()` des Moduls `webbrowser` kann den Browser starten und darin den angegebenen URL öffnen. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> import webbrowser  
>>> webbrowser.open('http://inventwithpython.com/')
```

Daraufhin wird in einem Browsetab der URL `http://inventwithpython.com/` geöffnet. Das ist so ziemlich das Einzige, was das Modul `webbrowser` tun kann. Allerdings macht die Funktion `open()` doch einige sehr interessante Dinge möglich. Wenn Sie für eine Postanschrift die Google-Maps-Karte anzeigen lassen wollen, können Sie ein einfaches Skript schreiben, das den Browser automatisch aufruft und darin anhand der Informationen in der Zwischenablage eine Karte öffnet. Dann müssen Sie nur noch die Anschrift in die Zwischenablage kopieren und das Skript ausführen, und schon wird die richtige Karte für Sie geöffnet.

Das Programm soll also Folgendes tun:

- Die Anschrift aus den Befehlszeilenargumenten oder der Zwischenablage beziehen
- Im Webbrower die Google-Maps-Seite für diese Anschrift öffnen

Dazu muss der Code folgende Aufgaben ausführen:

- Die Befehlszeilenargumente in `sys.argv` lesen
- Den Inhalt der Zwischenablage lesen
- Die Funktion `webbrowser.open()` aufrufen, um den Browser zu öffnen

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das neue Programm als `mapIt.py`.

## Schritt 1: Den URL herausfinden

Richten Sie *mapIt.py* nach der Anleitung in Anhang B so ein, dass es statt auf die Zwischenablage auf die Befehlszeilenargumente zurückgreift, wenn Sie es wie folgt an der Befehlszeile aufrufen:

```
C:\> mapit 870 Valencia St, San Francisco, CA 94110
```

Sind keine Befehlszeilenargumente vorhanden, nutzt das Programm den Inhalt der Zwischenablage.

Als Erstes müssen wir wissen, wie der URL für eine gegebene Straßenadresse aussehen muss. Wenn Sie im Browser <http://maps.google.com/> öffnen und nach einer bestimmten Anschrift suchen, sieht der URL in der Adressleiste etwa wie folgt aus: <https://www.google.com/maps/place/870+Valencia+St/@37.7590311,-122.4215096,17z/data=!3m1!4b1!4m2!3m1!1s0x808f7e3dad07a37:0xc86b0b2bb93b73d8>.

Die Anschrift steckt in diesem URL, aber daneben gibt es auch eine ganze Menge zusätzlichen Text. Auf Websites werden häufig Zusatzangaben zu URLs hinzugefügt, um die Bewegungen der Besucher zu verfolgen oder um die Seiten zu personalisieren. Sie können aber auch eine einfachere Version wie <https://www.google.com/maps/place/870+Valencia+St+San+Francisco+CA/> in der Adressleiste angeben und es wird trotzdem die richtige Seite angezeigt. Daher können Sie auch Ihr Programm so einrichten, dass es '<https://www.google.com/maps/place/adressstring>' im Browser öffnet (wobei *adressstring* die Anschrift ist, zu der Sie die Karte sehen möchten).

## Schritt 2: Befehlszeilenargumente verarbeiten

Geben Sie folgenden Code ein:

```
#! python3
# mapIt.py - Öffnet im Browser eine Karte zu der Anschrift in der
# Befehlszeile oder in der Zwischenablage

import webbrowser, sys
if len(sys.argv) > 1:
    # Ruft die Adresse von der Befehlszeile ab
    address = ' '.join(sys.argv[1:])

    # TODO: Adresse von der Zwischenablage abrufen
```

Nach der `#!`-Zeile müssen Sie das Modul `webbrowser` importieren, um den Browser öffnen zu können, und das Modul `sys`, um Befehlszeilenargumente zu lesen. In der Variablen `sys.argv` ist eine Liste gespeichert, die den Dateinamen des Programms und die Befehlszeilenargumente enthält. Steht in dieser Liste mehr als nur der Dateiname, dann wird `len(sys.argv)` zu einem Integer größer 1 ausgewertet. Daraus können Sie schließen, dass in der Tat Befehlszeilenargumente übergeben wurden.

Gewöhnlich dienen Leerzeichen dazu, mehrere Befehlszeilenargumente zu trennen, aber in diesem Programm soll die Gesamtmenge aller Argumente als ein einziger String aufgefasst werden. Da `sys.argv` eine Liste von Strings ist, können Sie diese Variable an die Methode `join()` übergeben, die einen einzelnen Stringwert zurückgibt. Da der Programmname in diesem String jedoch nicht enthalten sein soll, übergeben Sie nicht `sys.argv`, sondern `sys.argv[1:]`, um das erste Element des Arrays zu entfernen. Der String, den der Ausdruck letztlich ergibt, wird in der Variablen `address` gespeichert.

Nehmen Sie an, Sie führen das Programm wie folgt an der Befehlszeile aus:

```
mapit 870 Valencia St, San Francisco, CA 94110
```

Die Variable `sys.argv` enthält dann folgenden Listenwert:

```
['mapIt.py', '870', 'Valencia', 'St, ', 'San', 'Francisco, ', 'CA', '94110']
```

In der Variablen `address` steht dann der String `'870 Valencia St, San Francisco, CA 94110'`.

### Schritt 3: Den Inhalt der Zwischenablage verarbeiten und den Browser starten

Ergänzen Sie den Code wie folgt:

```
#! python3
# mapIt.py - Öffnet im Browser eine Karte zu der Anschrift in der
# Befehlszeile oder in der Zwischenablage

import webbrowser, sys, pyperclip
if len(sys.argv) > 1:
    # Ruft die Adresse von der Befehlszeile ab
    address = ' '.join(sys.argv[1:])
else:
    # Ruft die Adresse aus der Zwischenablage ab
    address = pyperclip.paste()

webbrowser.open('https://www.google.com/maps/place/' + address)
```

Wenn keine Befehlszeilenargumente vorhanden sind, geht das Programm davon aus, dass die Anschrift in der Zwischenablage steht. Den Inhalt der Zwischenablage

können Sie mit `pyperclip.paste()` abrufen und in der Variablen `address` speichern. Um den Browser mit dem URL für die entsprechende Google-Maps-Seite zu öffnen, rufen Sie `webbrowser.open()` auf.

Programme können Ihnen enorme Aufgaben abnehmen und dadurch stundenlange Arbeit ersparen. Es kann aber auch ebenso befriedigend sein, ein Programm zu schreiben, das Ihnen bei jeder Ausführung einer bestimmten Aufgabe nur ein paar Sekunden spart – so wie dieses. Tabelle 11–1 zeigt einen Vergleich der erforderlichen Schritte, um eine Karte mit und ohne *mapIt.py* aufzurufen.

Manuell	Mit <i>mapIt.py</i>
Anschrift markieren	Anschrift markieren
Adresse kopieren	Adresse kopieren
Webbrowser öffnen	<i>mapIt.py</i> ausführen
<code>http://maps.google.com/</code> aufrufen	
In das Adressfeld klicken	
Anschrift einfügen	
Eingabetaste drücken	

**Tab. 11–1** Karten mit und ohne *mapIt.py* aufrufen

Erkennen Sie, wie *mapIt.py* die Aufgabe vereinfacht?

### Vorschläge für ähnliche Programme

Wenn Sie einen URL haben, nimmt Ihnen das Modul `webbrowser` die Mühe ab, den Browser selbst zu öffnen und darin die gewünschte Website aufzurufen. Damit können Sie auch Programme schreiben, die folgende Aufgaben ausführen:

- Alle Links auf einer Seite in eigenen Browsetabs öffnen
- Den Browser öffnen und den URL für die lokale Wettervorhersage aufrufen
- Alle Social-Network-Sites öffnen, auf denen Sie regelmäßig nachschauen

## Dateien mithilfe des Moduls `requests` aus dem Web herunterladen

Mit dem Modul `requests` können Sie ganz einfach Dateien aus dem Web herunterladen, ohne sich um komplizierte Dinge wie Netzwerkfehler, Verbindungsprobleme und Datenkomprimierung zu kümmern. Da dieses Modul nicht im Lieferumfang von Python enthalten ist, müssen Sie es zunächst installieren. Führen Sie an der Befehlszeile `pip install requests` aus. (Eine ausführliche Anleitung zur Installation von Drittanbietermodulen finden Sie in Anhang A.)

Dieses Modul wurde geschrieben, da das Python-eigene Modul `urllib2` viel zu kompliziert zur Benutzung ist. Am besten nehmen Sie einen dicken Edding und schwärzen diesen ganzen Absatz. Vergessen Sie, dass ich `urllib2` jemals erwähnt habe. Wenn Sie irgendetwas aus dem Web herunterladen müssen, verwenden Sie einfach `requests`.

Um zu prüfen, ob das Modul `requests` korrekt installiert wurde, geben Sie Folgendes in die interaktive Shell ein:

```
>>> import requests
```

Wenn keine Fehlermeldungen erscheinen, wurde das Modul richtig installiert.

### Eine Webseite mit der Funktion `requests.get()` herunterladen

Die Funktion `requests.get()` nimmt einen String mit dem URL der herunterzuladenden Datei entgegen. Wenn Sie `type()` für den Rückgabewert dieser Funktion aufrufen, können Sie erkennen, dass es sich dabei um ein Response-Objekt handelt. Darin ist die Antwort enthalten, die der Webserver auf Ihre Anforderung gegeben hat. Response-Objekte werde ich in Kürze etwas ausführlicher erklären, aber zunächst geben Sie Folgendes in die interaktive Shell ein, während Ihr Computer noch mit dem Internet verbunden ist:

```
>>> import requests
>>> res = requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.txt')
❶
>>> type(res)
<class 'requests.models.Response'>
>>> res.status_code == requests.codes.ok    ❷
True
>>> len(res.text)
178981
>>> print(res.text[:250])
The Project Gutenberg EBook of Romeo and Juliet, by William Shakespeare
```

This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License.

Der URL führt zu einer Webseite mit dem gesamten Text des Schauspiels *Romeo und Julia*, zur Verfügung gestellt von Projekt Gutenberg (❶). Ob die Anforderung dieser Webseite erfolgreich verlaufen ist, können Sie überprüfen, indem Sie sich das Attribut `status_code` des Response-Objekts ansehen. Wenn es den Wert `requests.codes.ok` hat, dann ist alles in Ordnung (❷). (Der Statuscode für OK in HTTP ist

übrigens 200. Wahrscheinlich sind Sie bereits mit dem Statuscode 404 für »nicht gefunden« vertraut.)

Bei einer erfolgreichen Anforderung wird die heruntergeladene Webseite als String in der Variablen `text` des Response-Objekts gespeichert. Diese Variable enthält nun einen riesigen String mit dem vollständigen Theaterstück. Wenn Sie `len(res.text)` aufrufen, können Sie erkennen, dass dieser String mehr als 178.000 Zeichen lang ist. Mit `print(res.text[:250])` werden schließlich die ersten 250 Zeichen dieses Strings ausgegeben.

## Nach Fehlern suchen

Das Response-Objekt verfügt über das Attribut `status_code`, das Sie mit `requests.codes.ok` überprüfen können, um herauszufinden, ob der Download erfolgreich verlaufen ist. Eine einfachere Möglichkeit dazu besteht darin, die Methode `raise_for_status()` für das Response-Objekt aufzurufen. Dadurch wird eine Ausnahme ausgelöst, wenn beim Download der Datei ein Fehler aufgetreten ist. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> res = requests.get('http://inventwithpython.com/page_that_does_not_
exist')
>>> res.raise_for_status()
Traceback (most recent call last):
  File "<pyshell#138>", line 1, in <module>
    res.raise_for_status()
  File "C:\Python34\lib\site-packages\requests\models.py", line 773, in
    raise_for_status
      raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found
```

Die Methode `raise_for_status()` stellt eine gute Möglichkeit dar, um dafür zu sorgen, dass das Programm anhält, wenn der Download fehlerhaft war. Das ist eine gute Sache, denn schließlich sollte das Programm möglichst früh anhalten, wenn ein unerwarteter Fehler auftritt. Falls ein fehlgeschlagener Download die weitere Nutzung des Programms jedoch nicht komplett sinnlos macht, können Sie die von `raise_for_status()` ausgelöste Ausnahme auch mit `try-` und `except-` Anweisungen abfangen, damit das Programm nicht abstürzt.

```
import requests
res = requests.get('http://inventwithpython.com/page_that_does_not_exist')
try:
    res.raise_for_status()
except Exception as exc:
    print('There was a problem: %s' % (exc))
```

Aufgrund des Aufrufs von `raise_for_status()` gibt das Programm Folgendes aus:

```
There was a problem: 404 Client Error: Not Found
```

Rufen Sie nach `requests.get()` immer `raise_for_status()` auf, denn schließlich wollen Sie sicher sein, dass der Download auch tatsächlich funktioniert hat, bevor Sie mit dem Programm fortfahren.

## Heruntergeladene Dateien auf der Festplatte speichern

Die heruntergeladene Webseite können Sie anschließend mit der Standardfunktion `open()` und der Methode `write()` als Datei auf Ihrer Festplatte speichern. Es gibt jedoch einige Unterschiede zu den Fällen, in denen Sie diese Funktionen früher eingesetzt haben. Erstens müssen Sie die Datei im *binären Schreibmodus* öffnen, indem Sie 'wb' als zweites Argument an `open()` übergeben. Selbst wenn es sich um eine reine Textdatei handelt (wie den Text von *Romeo und Julia*, den Sie im vorherigen Beispiel heruntergeladen haben), müssen Sie Binär- statt Textdaten schreiben, um die *Unicode-Codierung* des Textes beizubehalten.

### Unicode-Codierung

Eine Erörterung von Unicode würde den Rahmen dieses Buchs sprengen, aber auf den folgenden Webseiten können Sie mehr darüber erfahren:

- Joel on Software: »The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)«, <http://www.joelonsoftware.com/articles/Unicode.html>
- Pragmatic Unicode: <http://nedbatchelder.com/text/unipain.html>

Um die Webseite in eine Datei zu schreiben, verwenden Sie eine `for`-Schleife mit der Methode `iter_content()` des Response-Objekts:

```
>>> import requests
>>> res = requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.txt')
>>> res.raise_for_status()
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
    playFile.write(chunk)

100000
78981
>>> playFile.close()
```

Die Methode `iter_content()` gibt bei jedem Schleifendurchlauf »Häppchen« des Inhalts zurück. Jedes dieser Häppchen ist vom Datentyp `byte`, wobei Sie angeben können, wie viele Bytes jeder dieser Abschnitte enthalten soll. Eine Größe von 100.000 Byte ist gewöhnlich gut geeignet, weshalb Sie 100000 als Argument an die Methode übergeben.

Anschließend ist im aktuellen Arbeitsverzeichnis die Datei *RomeoAndJuliet.txt* vorhanden. Auf der Website hatte die Datei den Namen *pg1112.txt*, aber für die Datei auf Ihrer Festplatte können Sie einen anderen Namen vergeben. Das Modul `requests` kümmert sich nur darum, die Inhalte von Webseiten herunterzuladen. Danach sieht Ihr Programm in dieser Seite nur noch Daten, mit denen es beliebig umgehen kann. Wenn Sie nach dem Herunterladen die Internetverbindung verlieren, befinden sich die Seitendaten immer noch auf Ihrem Computer.

Die Methode `write()` gibt die Anzahl der Bytes zurück, die in die Datei geschrieben wurden. In unserem Beispiel waren das 100.000 Byte beim ersten Durchlauf, sodass noch 78.981 Byte übrig bleiben.

Der vollständige Ablauf zum Herunterladen und Speichern der Datei sieht wie folgt aus:

1. Rufen Sie `requests.get()` auf, um die Datei herunterzuladen.
2. Rufen Sie `open()` mit 'wb' auf, um eine neue Datei anzulegen und im binären Schreibmodus zu öffnen.
3. Durchlaufen Sie die Schleife mithilfe der Methode `iter_content()` des `Response`-Objekts.
4. Rufen Sie bei jeder Iteration `write()` auf, um Inhalte in die Datei zu schreiben.
5. Rufen Sie `close()` auf, um die Datei zu schließen.

Mehr ist zur Nutzung des Moduls `requests` nicht zu tun! Im Vergleich mit dem üblichen Arbeitsablauf von `open()`, `write()` und `close()` für Textdateien wirken die `for`-Schleife und `iter_content()` etwas kompliziert, aber diese Vorgehensweise dient nur dazu, dass das `requests`-Modul beim Herunterladen riesiger Dateien nicht zu viel Arbeitsspeicher auf einmal mit Beschlag belegt. Die anderen Merkmale von `requests` können Sie auf <http://requests.readthedocs.org/> kennenlernen.

## HTML

Bevor wir uns eingehender mit Webseiten beschäftigen, müssen Sie einige Grundlagen von HTML kennenlernen. Außerdem erfahren Sie, wie Sie auf die Entwicklertools Ihres Browsers zugreifen. Damit wird das »Schürfen« nach Informationen im Web viel einfacher.

## Quellen zu HTML

*HyperText Markup Language* (HTML) ist das Format, in dem Webseiten geschrieben werden. In diesem Kapitel wird vorausgesetzt, dass Sie Grundkenntnisse in HTML haben. Wenn Sie einen Grundkurs in HTML brauchen, empfehle ich folgende Websites:

- <http://htmldog.com/guides/html/beginner/>
- <http://www.codecademy.com/tracks/web/>
- <https://developer.mozilla.org/en-US/learn/html/>

## Ein kleiner Auffrischungskurs

Für den Fall, dass es schon eine Weile her ist, dass Sie sich zuletzt mit HTML beschäftigt haben, finden Sie in diesem Abschnitt einen kleinen Überblick über die Grundlagen. Eine HTML-Datei ist eine einfache Textdatei mit der Endung *.html*. In einer solchen Datei stehen die einzelnen Textteile zwischen *Tags*, bei denen es sich um Schlüsselwörter in spitzen Klammern handelt. Sie teilen dem Browser mit, wie der eingeschlossene Text jeweils formatiert werden soll. Ein öffnendes und ein schließendes Tag bilden zusammen mit dem dazwischenstehenden Text (dem *inneren HTML*) ein *Element*. Der folgende HTML-Beispielcode zeigt *Hello world!* im Browser an, wobei *Hello* fett dargestellt wird:

```
<strong>Hello</strong> world!
```

In einem Browser sieht das Ergebnis so aus wie in Abb. 11–1.



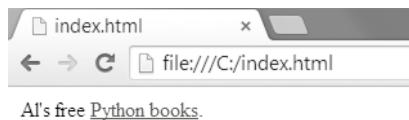
**Abb. 11–1** Hello world! im Browser

Das öffnende Tag `<strong>` weist den Browser an, den Text zwischen diesem und dem schließenden Tag `</strong>` fett darzustellen. Das schließende Tag `</strong>` markiert das Ende der Fettstellung.

In HTML gibt es viele verschiedene Tags. Einige weisen zusätzliche Eigenschaften in Form sogenannter *Attribute* auf, die ebenfalls in den spitzen Klammern angegeben werden. Beispielsweise sorgt `<a>` dafür, dass der Text zwischen den Tags als Link formatiert werden soll. Der URL, zu dem dieser Link führen soll, wird im Attribut `href` angegeben:

```
All's free <a href="http://inventwithpython.com">Python books</a>.
```

Im Browser wird dieser HTML-Code wie in Abb. 11–2 dargestellt.

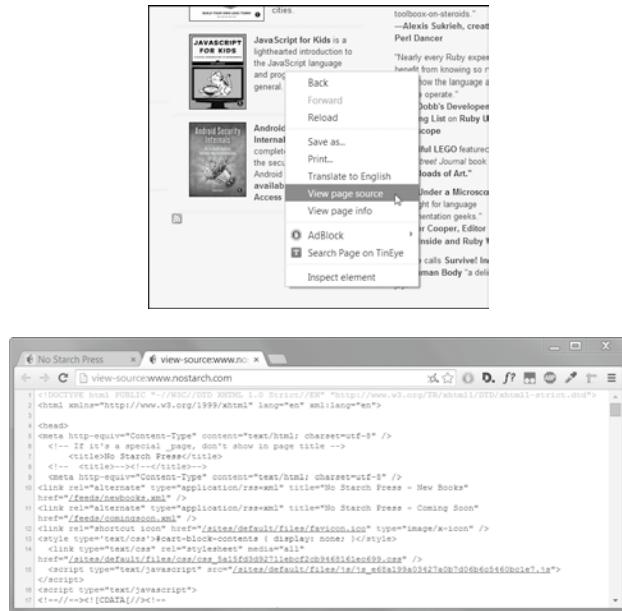


**Abb. 11–2** Ein Link in der Darstellung im Browser

Manche Elemente verfügen auch über ein `id`-Attribut, mit dem sie auf der Seite eindeutig identifiziert werden können. Da Sie in Ihren Programmen sehr häufig anhand des `id`-Attributs nach einem bestimmten Element suchen müssen, gehört die Verwendung des Entwicklerools des Browsers zu den häufigsten Aufgaben beim Schreiben von Web-Scraping-Programmen.

### Den HTML-Quellcode einer Webseite einsehen

Es bleibt nicht aus, dass Sie sich den HTML-Quellcode der Webseiten ansehen müssen, mit denen Ihre Programme arbeiten sollen. Dazu rechtsklicken Sie im Browser auf die betreffende Webseite (auf OS X klicken Sie bei gedrückter `[ctrl]`-Taste) und wählen *Quellcode anzeigen* oder *Seitenquelltext anzeigen* o. Ä. (siehe Abb. 11–3). Was Sie dann sehen, ist der Text, den der Browser empfängt. Der HTML-Code teilt dem Browser mit, wie er die Webseite darstellen soll.

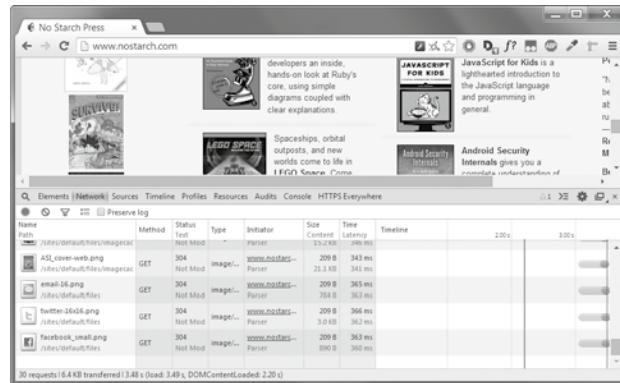


**Abb. 11–3** Den Quellcode einer Webseite anzeigen

Ich empfehle Ihnen sehr, sich den HTML-Quelltext Ihrer Lieblingswebsites anzusehen. Es macht nichts, wenn Sie dabei nicht alles verstehen. Um einfache Web-Scraping-Programme zu schreiben, müssen Sie kein Meister in HTML sein, denn schließlich wollen Sie damit ja keine eigene Website schreiben. Sie müssen nur genug wissen, um die gewünschten Daten aus einer vorhandenen Website herauspicken zu können.

## Die Entwicklertools des Browsers öffnen

Den HTML-Code einer Webseite können Sie sich nicht nur in der Quelltextansicht anschauen, sondern auch in den Entwicklertools des Browsers. In Chrome und Internet Explorer sind sie bereits installiert und können mit **F12** eingeblendet und wieder ausgeblendet werden (siehe Abb. 11–4). In Chrome können Sie sie außerdem öffnen, indem Sie auf die Schaltfläche mit den drei waagerechten Balken klicken und *Weitere Tools > Entwicklertools* wählen. Unter OS X erscheinen die Entwicklertools, wenn Sie **cmd** + **alt** + **I** drücken.



**Abb. 11–4** Die Entwicklertools in Chrome

In Firefox können Sie den Inspektor der Webentwicklertools aufrufen, indem Sie unter Windows und Linux **Strg** + **Umschalt** + **C** drücken bzw. **cmd** + **alt** + **C** unter OS X. Das Layout ist fast identisch mit dem der Entwicklertools von Chrome.

In Safari öffnen Sie das Menü *Einstellungen* und aktivieren dann im Bereich *Erweitert* die Option *Menü "Entwickler" in der Menüleiste anzeigen*. Danach können Sie die Entwicklertools mit **cmd** + **alt** + **I** einblenden.

Wenn diese Einrichtung erledigt ist, können Sie auf einen beliebigen Teil einer Webseite rechtsklicken und *Element untersuchen* auswählen. Daraufhin wird der für diesen Teil der Webseite verantwortliche HTML-Code angezeigt. Das ist sehr nützlich, wenn Sie in Ihren Web-Scraping-Programmen den HTML-Text durchforsten müssen.

### Keine regulären Ausdrücke zur Suche in HTML

Die Aufgabe, einen bestimmten HTML-Code in einem String zu finden, scheint wie für reguläre Ausdrücke gemacht zu sein. Allerdings rate ich Ihnen davon ab. Es gibt sehr viele unterschiedliche Formatierungsmöglichkeiten für HTML, die alle gültig sind, und der Versuch, all diese Varianten in einem regulären Ausdruck abzudecken, kann sehr mühselig und fehleranfällig sein. Weniger Fehler sind zu erwarten, wenn Sie ein Modul verwenden, das eigens für die HTML-Analyse gedacht ist, z. B. BeautifulSoup.

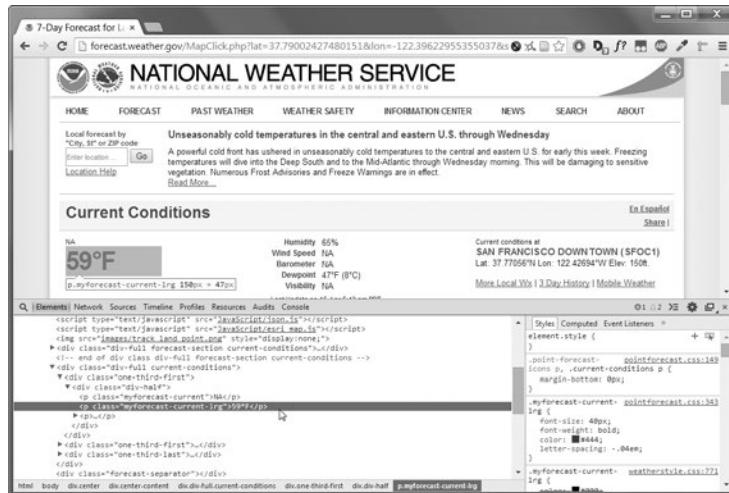
Eine ausgiebige Argumentation dafür, keine regulären Ausdrücke zur Analyse von HTML zu verwenden, finden Sie auf <http://stackoverflow.com/a/1732454/1893164/>.

## HTML-Elemente mithilfe der Entwicklertools finden

Nachdem ein Programm eine Webseite mithilfe des Moduls `requests` heruntergeladen kann, liegt der gesamte HTML-Inhalt dieser Seite als ein einziger Stringwert vor. Jetzt müssen Sie herausfinden, welcher Teil dieses HTML-Inhalts den Informationen auf der Webseite entspricht, an denen Sie interessiert sind.

Hier kommen die Entwicklertools des Browsers ins Spiel. Nehmen wir an, Sie wollen ein Programm schreiben, das die Daten der Wettervorhersage von <http://weather.gov/> abruft. Bevor Sie anfangen, Code zu schreiben, müssen Sie jedoch erst ein wenig recherchieren. Wenn Sie die Website aufsuchen und dort nach der (US-)Postleitzahl 94105 suchen, werden Sie zu einer Seite geleitet, die die Wettervorhersage für den betreffenden Bereich anzeigt.

Stellen Sie sich vor, Sie wollen die Temperaturinformationen für dieses Gebiet abrufen. Rechtsklicken Sie auf die Temperaturangabe auf der Seite (auf OS X klicken Sie bei gedrückter `ctrl`-Taste) und wählen Sie in dem daraufhin eingeblendeten Kontextmenü *Element untersuchen*. Dadurch wird das Fenster mit den Entwicklerwerkzeugen geöffnet, in dem Sie den HTML-Code für den betreffenden Bereich der Seite sehen (siehe Abb. 11–5).



**Abb. 11–5** Das Element mit der Temperaturangabe in den Entwicklertools untersuchen

In den Entwicklertools können Sie erkennen, dass das HTML-Element für die Temperaturangabe auf der Webseite `<p class="myforecast-current-1rg">57°F</p>` ist. Das ist genau das, wonach Sie suchen! Die Temperaturinformationen stehen in einem `<p>`-Element der Klasse `myforecast-current-1rg`. Nachdem Sie nun wissen, wonach Sie Ausschau halten müssen, können Sie das Modul Beautiful Soup einsetzen, um die Informationen in dem heruntergeladenen String zu finden.

## HTML mit dem Modul BeautifulSoup durchsuchen

Das Modul BeautifulSoup dient dazu, Informationen aus einer HTML-Seite zu entnehmen (und ist für diesen Zweck besser geeignet als reguläre Ausdrücke). Der Modulname lautet `bs4` (was für BeautifulSoup Version 4 steht). Zur Installation müssen Sie an der Befehlszeile `pip install beautifulsoup4` ausführen. (Eine Anleitung zur Installation von Drittanbietermodulen erhalten Sie in Anhang A.) Für die Installation müssen Sie zwar die Bezeichnung `beautifulsoup4` verwenden, aber zum Import von BeautifulSoup schreiben Sie `import bs4`.

In den Beautiful-Soup-Beispielen in diesem Kapitel wird eine HTML-Datei auf Ihrer Festplatte *geparst* (d. h., sie wird analysiert und ihre einzelnen Bestandteile werden identifiziert). Geben Sie im Dateieditor den folgenden Code ein und speichern Sie ihn als `example.html`. Alternativ können Sie diese Datei auch von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.

```
<!-- Dies ist die Beispieldatei example.html. -->

<html><head><title>The Website Title</title></head>
<body>
<p>Download my <strong>Python</strong> book from <a href="http://
    inventwithpython.com">my website</a>.</p>
<p class="slogan">Learn Python the easy way!</p>
<p>By <span id="author">Al Sweigart</span></p>
</body></html>
```

Wie Sie sehen, enthält schon eine ganz einfache HTML-Datei viele verschiedene Tags und Attribute. Bei anspruchsvolleren Websites kann die Sache ziemlich verwirrend werden. Zum Glück macht BeautifulSoup die Arbeit mit HTML sehr einfach.

### Ein BeautifulSoup-Objekt aus dem HTML-Text erstellen

Wenn Sie die Funktion `bs4.BeautifulSoup()` mit einem String aufrufen, der den zu passenden HTML-Text enthält, gibt sie ein BeautifulSoup-Objekt zurück. Das können Sie wie folgt in der interaktiven Shell ausprobieren, wobei Ihr Computer mit dem Internet verbunden sein muss:

```
>>> import requests, bs4
>>> res = requests.get('http://nostarch.com')
>>> res.raise_for_status()
>>> noStarchSoup = bs4.BeautifulSoup(res.text)
>>> type(noStarchSoup)
<class 'bs4.BeautifulSoup'>
```

Hier wird mithilfe von `requests.get()` die Hauptseite der Website von No Starch Press heruntergeladen und das Attribut `text` der Antwort an `bs4.BeautifulSoup()` übergeben. Das von dieser Funktion zurückgegebene `BeautifulSoup`-Objekt wiederum wird in der Variablen `noStarchSoup` gespeichert.

Sie können auch eine HTML-Datei von Ihrer Festplatte laden, indem Sie `bs4.BeautifulSoup()` das entsprechende `File`-Objekt übergeben. Das können Sie wie folgt in der interaktiven Shell ausprobieren, wobei sich die Datei `example.html` jedoch in Ihrem aktuellen Arbeitsverzeichnis befinden muss:

```
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile)
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

Sobald Ihnen ein `BeautifulSoup`-Objekt vorliegt, können Sie dessen Methoden nutzen, um die einzelnen Teile des HTML-Dokuments zu finden.

### Elemente mit der Methode `select()` finden

Um aus einem `BeautifulSoup`-Element ein Webseitenelement abzurufen, verwenden Sie die Methode `select()` und übergeben ihr den String eines *CSS-Selektors* für das gesuchte Element. Selektoren ähneln regulären Ausdrücken darin, dass auch sie ein Muster für die gesuchten Elemente angeben. Allerdings findet die Suche hier auf HTML-Seiten statt und nicht in allgemeinen Textstrings.

Eine ausführliche Beschreibung der Syntax von CSS-Selektoren würde den Rahmen dieses Buchs sprengen. Ein gutes Tutorial finden Sie unter den Ressourcen auf der Seite [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) und im Folgenden gebe ich Ihnen eine kurze Einführung. Tabelle 11–2 zeigt Beispiele für die gebräuchlichsten CSS-Selektormuster.

Selektor	Bedeutung
<code>soup.select('div')</code>	Findet alle <code>div</code> -Elemente
<code>soup.select('#author')</code>	Findet alle Elemente mit dem <code>id</code> -Attribut <code>author</code>
<code>soup.select('.notice')</code>	Findet alle Elemente, bei denen das CSS-Attribut <code>class</code> den Wert <code>notice</code> hat
<code>soup.select('div span')</code>	Findet alle <code>&lt;span&gt;</code> -Elemente innerhalb eines <code>&lt;div&gt;</code> -Elements
<code>soup.select('div &gt; span')</code>	Findet alle <code>&lt;span&gt;</code> -Elemente, die <i>unmittelbar</i> in einem <code>&lt;div&gt;</code> -Element verschachtelt sind, also ohne weitere Ebenen dazwischen

Selektor	Bedeutung
soup.select('input[name]')	Findet alle <input>-Elemente, die über ein name-Attribut mit beliebigem Wert verfügen
soup.select('input[-type="button"]')	Findet alle <input>-Elemente, die über ein type-Attribut mit dem Wert button verfügen

**Tab. 11–2** Beispiele für CSS-Selektoren

Die einzelnen Selektormuster können zu anspruchsvollen Suchvorgängen kombiniert werden. Beispielsweise findet `soup.select('p #author')` alle Elemente, die sich innerhalb eines `<p>`-Elements befinden und deren id-Attribut den Wert author hat.

Die Methode `select()` gibt eine Liste von Tag-Objekten zurück, wobei jedes dieser Objekte für ein übereinstimmendes Element in dem HTML-Text des BeautifulSoup-Objekts steht. Um den kompletten Text der Elemente zu sehen, übergeben Sie die Tag-Werte an die Funktion `str()`. Außerdem verfügen Tag-Werte über das Attribut `attrs`, das alle HTML-Attribute des betreffenden Tags in Form eines Dictionarys festhält. Zum Ausprobieren greifen wir wieder auf die Datei *example.html* aus unserem früheren Beispiel zurück und geben Folgendes in die interaktive Shell ein:

```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read())
>>> elems = exampleSoup.select('#author')
>>> type(elems)
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> elems[0].getText()
'Al Sweigart'
>>> str(elems[0])
'<span id="author">Al Sweigart</span>'
>>> elems[0].attrs
{'id': 'author'}
```

Dieser Code entnimmt dem HTML-Beispieltext das Element mit `id="author"`. Die Methode `select('#author')` gibt diese Elemente als Liste von Tag-Objekten zurück, die wir in der Variablen `elems` speichern. Mit `len(elems)` bringen wir in Erfahrung, dass diese Liste nur ein einziges Objekt dieser Art entfällt – es wurde also nur eine Übereinstimmung gefunden. Wenn wir `getText()` für dieses Objekt aufrufen, erhalten wir den Text, also den inneren HTML-Code des Elements zurück. Dies ist

der Inhalt zwischen dem öffnenden und dem schließenden Tag, in unserem Fall also 'Al Sweigart'.

Übergeben wir das Tag-Objekt an `str()`, erhalten wir einen String mit dem kompletten Element, also dem öffnenden und dem schließenden Tag sowie dem darin eingeschlossenen Text. Mit `attrs` können wir schließlich ein Dictionary mit den Attributen des Elements abrufen. In unserem Fall ist das lediglich das Attribut `id` mit dem Wert 'author'.

Im folgenden Beispiel rufen Sie alle `<p>`-Elemente aus dem BeautifulSoup-Soup-Objekt ab:

```
>>> pElems = exampleSoup.select('p')
>>> str(pElems[0])
'<p>Download my <strong>Python</strong> book from <a href="http://
    inventwithpython.com">my website</a>.</p>'
>>> pElems[0].getText()
'Download my Python book from my website.'
>>> str(pElems[1])
'<p class="slogan">Learn Python the easy way!</p>'
>>> pElems[1].getText()
'Learn Python the easy way!'
>>> str(pElems[2])
'<p>By <span id="author">Al Sweigart</span></p>'
>>> pElems[2].getText()
'By Al Sweigart'
```

Diesmal enthält die von `select()` zurückgegebene Liste drei Übereinstimmungen, die wir in `pElems` speichern. Um sich die Strings der kompletten Elemente anzusehen, wenden Sie `str()` auf `pElems[0]`, `pElems[1]` und `pElems[1]` an. Die Funktion `getText()` gibt für die einzelnen Elemente dagegen nur den Text zwischen den Tags zurück.

## Daten aus den Attributen eines Elements abrufen

Mit der Methode `get()` für Tag-Objekte ist es ganz einfach, auf die Attributwerte eines Elements zuzugreifen. Wenn Sie dieser Methode den String eines Attributnamens übergeben, gibt sie den zugehörigen Attributwert zurück. Probieren Sie das wie folgt anhand von `example.html` in der interaktiven Shell aus:

```
>>> import bs4
>>> soup = bs4.BeautifulSoup(open('example.html'))
>>> spanElem = soup.select('span')[0]
>>> str(spanElem)
'<span id="author">Al Sweigart</span>'
>>> spanElem.get('id')
'author'
```

```
>>> spanElem.get('some_nonexistent_addr') == None
True
>>> spanElem.attrs
{'id': 'author'}
```

Hier suchen wir zunächst mit `select()` nach `<span>`-Elementen und speichern dann den ersten Fund in `spanElement`. Wenn wir `get()` den Attributnamen `'id'` übergeben, erhalten wir den Wert dieses Attributs zurück, nämlich `'author'`.

## Projekt: Google-Suche »Auf gut Glück«

Bei der Suche in Google schaue ich mir nicht nur ein Suchergebnis auf einmal an. Stattdessen klicke ich mit der mittleren Maustaste (oder dem Mausrad; Sie können auch bei gedrückter `Strg`-Taste drücken) auf eine Handvoll der ersten Links, um sie gleichzeitig in mehreren Browsetabs zu öffnen. Ich suche so oft in Google, dass diese Vorgehensweise schon ziemlich ermüdend ist. Es wäre schön, wenn ich einfach meinen Suchbegriff an der Befehlszeile eingeben könnte und mein Computer dann automatisch den Browser öffnet und die ersten Suchergebnisse in einer Reihe von Registerkarten darstellt. Dafür wollen wir nun ein Skript schreiben.

Das Programm erfüllt folgende Aufgaben:

- Die Suchbegriffe aus den Befehlszeilenargumenten entnehmen
- Die Seite mit den Suchergebnissen abrufen
- Für jedes Ergebnis einen Browsetab öffnen

Dazu muss der Code Folgendes tun:

- Die Befehlszeilenargumente in `sys.argv` lesen
- Die Suchergebnisseite mithilfe des Moduls `requests` abrufen
- Die Links zu den einzelnen Suchergebnissen finden
- Die Funktion `webbrowser.open()` aufrufen, um den Browser zu öffnen

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das Programm als `lucky.py`.

### Schritt 1: Die Befehlszeilenargumente abrufen und die Suchergebnisseite anfordern

Bevor Sie anfangen können, Code zu schreiben, müssen Sie zunächst den URL der Suchergebnisseite kennen. Wenn Sie eine Google-Suche durchführen und sich anschließend die Adressleiste in Ihrem Browser anschauen, sehen Sie dort den URL `https://www.google.com/search?q=SUCHBEGRIFF`. Mit dem Modul `requests`

können Sie diese Seite herunterladen, um anschließend mit Beautiful Soup den HTML-Text nach den Links zu den Suchergebnissen zu durchsuchen. Danach öffnen Sie diese Links mit dem Modul `webbrowser` in Browsetabs.

Geben Sie folgenden Code ein:

```
#! python3
# lucky.py - Öffnet mehrere Google-Suchergebnisse

import requests, sys, webbrowser, bs4

print('Googling...') # Zeigt beim Herunterladen der Suchergebnisseite
                     # eine Meldung an
res = requests.get('http://google.com/search?q=' + ' '.join(sys.argv[1:]))
res.raise_for_status()

# TODO: Links der obersten Suchergebnisse abrufen

# TODO: Für jedes Ergebnis einen Browsetab öffnen
```

Beim Start des Programms muss der Benutzer seine Suchbegriffe als Befehlszeilenargumente angeben. Diese Argumente werden als Liste von Strings in `sys.argv` gespeichert.

## Schritt 2: Alle Ergebnisse finden

Als Nächstes verwenden Sie Beautiful Soup, um die Links der obersten Suchergebnisse in dem heruntergeladenen HTML-Text zu finden. Wie aber wählen Sie den richtigen Selektor dafür aus? Sie können nicht einfach nach allen `<a>`-Tags suchen, da es in dem HTML-Text auch eine Menge anderer Links gibt, die uns nicht interessieren. Um einen Selektor zu finden, der nur die gewünschten Links herauspickt, untersuchen Sie eine Suchergebnisseite mit den Entwicklertools des Browsers.

Wenn Sie in Google beispielsweise nach *Beautiful Soup* gesucht haben und die Entwicklertools öffnen, werden Sie feststellen, dass die Linkelemente auf der Suchergebnisseite furchterlich kompliziert aussehen, etwa wie folgt: `<a href="/url?sa=t&amp;rct=j&amp;q=&amp;esrc=s&amp;source=web&amp;cd=1&amp;cad=rja&amp;uact=8&amp;ved=0CCgQFjAA&amp;url=http%3A%2F%2Fwww.crummy.com%2Fsoftware%2FBeautifulSoup%2F&amp;ei=LHBVU_XDD9KVyAShmyDwCw&amp;usg=AFQjCNHAxwplurFOBqg5cehWQEVKi-TuLQ&amp;sig2=sdZu6WV1B1VSDrwhtworMA" onmousedown="return rwt(this, '', '', '', '1', 'AFQjCNHAxwplurFOBqg5cehWQEVKi-TuLQ', 'sdZu6WV1B1VSDrwhtworMA', '0CCgQFjAA', '', '', event)" data-href="http://www.crummy.com/software/BeautifulSoup/><em>BeautifulSoup</em>: We called him Tortoise because he taught us.</a>`.

Es spielt jedoch keine Rolle, wie kompliziert die Links aufgebaut sind. Sie müssen nur ein Muster erkennen, das in allen Suchergebnislinks vorkommt. Die <a>-Elemente der Suchergebnisse allerdings weisen keine Merkmale auf, anhand derer man sie schnell von anderen <a>-Elementen unterscheiden könnte.

In der Nähe der <a>-Elemente finden Sie jedoch das Element <h3 class="r">. Die Klasse r scheint in dem ganzen HTML-Quelltext ausschließlich in den Links für Suchergebnisse verwendet zu werden. Wir müssen uns nicht damit beschäftigen, was die CSS-Klasse r ist und welche Gestaltung sie bewirkt, sondern verwenden Sie einfach als Kennzeichen für die <a>-Elemente, nach denen wir suchen. Erstellen Sie aus dem HTML-Text der heruntergeladenen Suchergebnisseite ein BeautifulSoup-Objekt und bestimmen Sie dann anhand des Selektors '.r' alle <a>-Elemente, die sich in einem Element mit der CSS-Klasse r befinden. Der Code dazu sieht wie folgt aus:

```
#! python3
# lucky.py - Öffnet mehrere Google-Suchergebnisse

import requests, sys, webbrowser, bs4

-- schnipp --

# Ruft die Links der obersten Suchergebnisse ab
soup = bs4.BeautifulSoup(res.text)

# Öffnet einen Browsetab für jedes Ergebnis
linkElems = soup.select('.r a')
```

### Schritt 3: Browsetabs für jedes Suchergebnis öffnen

Als Letztes müssen wir das Programm anweisen, die Ergebnisse in Browsetabs zu öffnen. Ergänzen Sie das Ende des Programms wie folgt:

```
#! python3
# lucky.py - Öffnet mehrere Google-Suchergebnisse

import requests, sys, webbrowser, bs4

-- schnipp --

# Öffnet einen Browsetab für jedes Ergebnis
linkElems = soup.select('.r a')
numOpen = min(5, len(linkElems))
for i in range(numOpen):
    webbrowser.open('http://google.com' + linkElems[i].get('href'))
```

Hier werden mithilfe des Moduls `webbrowser` die ersten fünf Suchergebnisse in neuen Tabs geöffnet. Es kann jedoch auch sein, dass weniger als fünf Suchergebnisse angezeigt wurden. Da der Aufruf von `soup.select()` eine Liste aller Elemente zurückgibt, die mit dem Selektor `'.r a'` übereinstimmen, können Sie dafür sorgen, dass die Anzahl der zu öffnenden Tabs entweder gleich 5 oder gleich der Länge der Liste ist (wobei jeweils die kleinere Anzahl gewählt wird).

Die integrierte Python-Funktion `min()` gibt das kleinste der ihr übergebenen Integer- oder Fließkommaargumente zurück. (Es gibt auch die Funktion `max()`, die das größte übergebene Argument zurückgibt.) Mit `min()` können Sie herausfinden, ob es weniger als fünf Links in der Liste gibt, und die Anzahl der zu öffnenden Links in der Variable `numOpen` speichern. Anschließend durchlaufen Sie eine for-Schleife mit `range(numOpen)`.

Bei jedem Durchlauf der Schleife öffnen Sie mit `webbrowser.open()` einen neuen Browertab. Beachten Sie, dass der Wert des `href`-Attributs in den zurückgegebenen `<a>`-Elementen nicht über das einleitende `http://google.com` verfügt, weshalb Sie diesen String mit dem String des Attributs verketten müssen.

Nun können Sie sich automatisch die ersten fünf Google-Treffer beispielsweise für die Suche nach *Python programming tutorials* öffnen lassen, indem Sie an der Befehlszeile einfach `lucky python programming tutorials` ausführen. (Wie Sie Programme auf Ihrem Betriebssystem ausführen, erfahren Sie in Anhang B.)

### Vorschläge für ähnliche Programme

Ein Vorteil von Browertabs besteht darin, dass Sie Links in neuen Tabs öffnen können, um sie sich später anzusehen. Ein Programm, das automatisch mehrere Links auf einmal öffnet, ist hilfreich für folgende Aufgaben:

- Alle Produktseiten nach einer Suche auf einer Shoppingwebsite wie Amazon öffnen
- Alle Links mit Rezensionen zu einem einzelnen Produkt öffnen
- Die Ergebnislinks für Fotos nach einer Suche auf einer Fotowebseite wie Flickr oder Imgur öffnen

### Projekt: Alle XKCD-Comics herunterladen

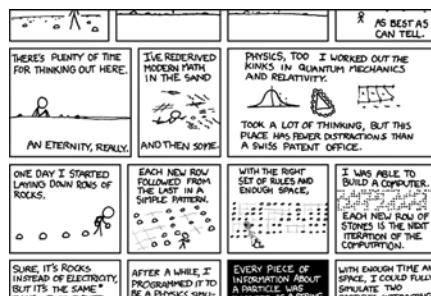
Auf der Startseite von Blogs und anderen regelmäßig aktualisierten Websites finden Sie gewöhnlich den aktuellen Eintrag sowie die Schaltfläche *Previous*, die zum vorhergehenden Eintrag führt. Dieser wiederum hat eine weitere *Previous*-Schaltfläche usw., sodass Sie sich vom aktuellen bis zum allerersten Eintrag auf der Website entlanghangeln können. Wenn Sie den Inhalt der Seite kopieren wollen,

um ihn offline zu lesen, könnten Sie auf diese Weise jede einzelne Seite aufrufen und speichern, doch das wäre eine ziemlich langweilige Aufgabe. Daher wollen wir ein Programm dafür schreiben.

Die Webseite des beliebten Webcomics XKCD (siehe Abb. 11–6) weist den angesprochenen Aufbau auf: Auf der Startseite von <http://xkcd.com/> finden Sie die Schaltfläche *Prev*, über die Sie zu den früheren Comics gelangen können. Jeden einzelnen Comic per Hand herunterzuladen, würde ewig dauern, aber Sie können ein Skript schreiben, das diese Aufgabe in wenigen Minuten erledigt.

Dieses Programm führt folgende Aufgaben aus:

- Die Startseite von XKCD laden
- Das Bild mit dem Comic auf dieser Seite speichern
- Dem Link zum vorherigen Comic folgen
- Den Vorgang wiederholen, bis der erste Comic erreicht ist



**Abb. 11–6** XKCD, »ein Webcomic mit Romantik, Sarkasmus, Mathematik und Sprache«

Der Code muss also Folgendes tun:

- Seiten mithilfe des Moduls `requests` herunterladen
- Den URL für das Bild des Comics auf der Seite mithilfe von `Beautiful Soup` finden
- Das Bild des Comics mit `iter_content()` herunterladen und auf der Festplatte speichern
- Den URL des Links zum vorherigen Comic finden und den bisherigen Ablauf wiederholen

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das Programm als `downloadXkcd.py`.

## Schritt 1: Den Aufbau des Programms festlegen

Wenn Sie die Entwicklertools des Browsers öffnen und damit die Elemente auf der XKCD-Startseite untersuchen, stellen Sie Folgendes fest:

- Der URL für die Bilddatei des Comics steht im href-Attribut eines <img>-Elements.
- Das <img>-Element befindet sich in einem <div id="comic">-Element.
- Die Schaltfläche *Prev* verfügt über das HTML-Attribut rel mit dem Wert prev.
- Die *Prev*-Schaltfläche des ersten Comics ist mit dem URL <http://xkcd.com/#> verknüpft, was bedeutet, dass es keine weiteren vorhergehenden Seiten gibt.

Geben Sie folgenden Code ein:

```
#! python3
# downloadXkcd.py - Lädt alle XKCD-Comics herunter

import requests, os, bs4

url = 'http://xkcd.com' # Start-URL
os.makedirs('xkcd', exist_ok=True) # Speichert die Comics in ./xkcd
while not url.endswith('#'):
    # TODO: Seite herunterladen

    # TODO: URL des Bildes für den Comic finden

    # TODO: Bild herunterladen

    # TODO: Bild in ./xkcd speichern

    # TODO: URL der Prev-Schaltfläche abrufen

    print('Done.')
```

Die Variable `url` hat zu Anfang den Wert '<http://xkcd.com>' und wird (in einer `for`-Schleife) wiederholt mit dem URL des *Prev*-Links auf der jeweils vorliegenden Seite aktualisiert. Bei jedem Schleifendurchlauf laden Sie den Comic von der Adresse herunter, die `url` gerade angibt. Wenn der Wert von `url` mit '#' endet, haben Sie das Ende der Schleife erreicht.

Die Bilddateien werden in den Ordner `xkcd` im aktuellen Arbeitsverzeichnis heruntergeladen, wobei der Aufruf von `os.makedirs()` dafür sorgt, dass dieser Ordner vorhanden ist. Das Schlüsselwort `exist_ok=True` verhindert, dass die Funktion aufgrund dieser Tatsache eine Ausnahme aufwirft. Der Rest des Codes besteht zurzeit noch aus Kommentaren, die die Struktur des Programms vorgeben.

## Schritt 2: Die Webseite herunterladen

Schreiben wir nun den Code, um die Seite herunterzuladen. Das Programm sieht danach wie folgt aus:

```
#! python3
# downloadXkcd.py - Lädt alle XKCD-Comics herunter

import requests, os, bs4

url = 'http://xkcd.com' # Start-URL
os.makedirs('xkcd', exist_ok=True) # Speichert die Comics in ./xkcd
while not url.endswith('#'):
    # Lädt die Seite herunter
    print('Downloading page %s...' % url)
    res = requests.get(url)
    res.raise_for_status()

    soup = bs4.BeautifulSoup(res.text)

    # TODO: URL des Bildes für den Comic finden

    # TODO: Bild herunterladen

    # TODO: Bild in ./xkcd speichern

    # TODO: URL der Prev-Schaltfläche abrufen

    print('Done.')
```

Als Erstes wird `url` ausgegeben, damit der Benutzer weiß, welcher URL als Nächstes heruntergeladen wird. Anschließend wird die Seite mithilfe der Funktion `requests.get()` des Moduls `requests` heruntergeladen. Wie immer rufen Sie hier sofort die Methode `raise_for_status()` des Response-Objekts auf, um eine Ausnahme auszulösen und das Programm zu beenden, wenn beim Download irgendetwas schiefgegangen ist. Ist alles in Ordnung, erstellen Sie ein `BeautifulSoup`-Objekt aus dem Text der heruntergeladenen Seite.

## Schritt 3: Das Bild des Comics finden und herunterladen

Ergänzen Sie den Code wie folgt:

```
#! python3
# downloadXkcd.py - Lädt alle XKCD-Comics herunter

import requests, os, bs4

-- schnipp --
```

```
# Sucht den URL mit dem Bild des Comics
comicElem = soup.select('#comic img')
if comicElem == []:
    print('Could not find comic image.')
else:
    comicUrl = comicElem[0].get('src')
    # Lädt das Bild herunter
    print('Downloading image %s...' % (comicUrl))
    res = requests.get(comicUrl)
    res.raise_for_status()

# TODO: Bild in ./xkcd speichern

# TODO: URL der Prev-Schaltfläche abrufen

print('Done.')
```

Bei der Untersuchung der XKCD-Startseite mit den Entwicklertools haben Sie festgestellt, dass sich das `<img>`-Element mit dem Bild für den Comic in einem `<div>`-Element mit dem id-Wert `comic` befindet. Daher können Sie mit dem Selektor `'#comic img'` die gewünschten `<img>`-Elemente aus dem BeautifulSoup-Objekt abrufen.

Es gibt einige wenige XKCD-Seiten mit besonderen Inhalten, bei denen es sich nicht um einfache Bilddateien handelt. Das ist kein Problem; diese Einträge überspringen wir einfach. Wenn der Selektor keine Elemente findet, gibt `soup.select('#comic img')` eine leere Liste zurück. Das Programm gibt dann einfach eine Fehlermeldung aus, ohne irgendetwas herunterzuladen.

In allen anderen Fällen gibt der Selektor eine Liste mit einem `<img>`-Element zurück. Um die Bilddatei des Comics herunterzuladen, rufen Sie das Attribut `src` dieses Elements ab und übergeben es an `requests.get()`.

#### Schritt 4: Das Bild speichern und den vorherigen Comic suchen

Ergänzen Sie den Code wie folgt:

```
#! python3
# downloadXkcd.py - Lädt alle XKCD-Comics herunter

import requests, os, bs4

-- schnipp --
```

```
# Speichert das Bild in ./xkcd
imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)),
                 'wb')
for chunk in res.iter_content(100000):
    imageFile.write(chunk)
imageFile.close()

# Ruft den URL der Prev-Schaltfläche ab
prevLink = soup.select('a[rel="prev"]')[0]
url = 'http://xkcd.com' + prevLink.get('href')

print('Done.')
```

Die Bilddatei des Comics befindet sich bereits in der Variablen `res` und muss nun in eine Datei auf der Festplatte geschrieben werden.

Dazu übergeben Sie `open()` einen Namen für die lokale Bilddatei. Der URL in `comicURL` hat einen Wert wie `'http://imgs.xkcd.com/comics/heartbleed_explanation.png'`, was wie ein Dateipfad aussieht. Daher können Sie `comicURL` an `os.path.basename()` übergeben, um den letzten Teil des URL zurückzugewinnen, hier `'heartbleed_explanation.png'`. Diese Bezeichnung wiederum können Sie als Dateinamen für die Speicherung auf der Festplatte verwenden. Um diesen Namen mit dem Namen Ihres `xkcd`-Ordners zu verknüpfen, setzen Sie `os.path.join()` ein, damit auf Windows Backslashes (`\`) und auf OS X und Linux Schrägstriche (`/`) als Pfadtrennzeichen verwendet werden. Damit haben Sie nun endlich den vollständigen Namen für die Datei, die Sie jetzt mit `open()` und `'wb'` im binären Schreibmodus öffnen können.

Um die mithilfe von `requests` heruntergeladenen Dateien zu speichern, müssen Sie, wie Sie weiter vorn in diesem Kapitel schon erfahren haben, in einer Schleife die Rückgabewerte der Methode `iter_content()` durchlaufen. Der Code in der `for`-Schleife schreibt nacheinander einzelne Abschnitte der Bilddaten (von je maximal 100.000 Byte) in die Datei und schließt diese, wenn der Vorgang abgeschlossen ist. Damit ist das Bild auf der Festplatte gespeichert.

Danach sucht der Selektor `'a[rel="prev"]'` nach dem `<a>`-Element, dessen `rel`-Attribut auf `prev` gesetzt ist. Aus dem `href`-Attribut dieses Elements können Sie dann den URL des vorherigen Comics entnehmen, der anschließend in `url` gespeichert wird. Daraufhin beginnt die `while`-Schleife erneut mit dem Download-Vorgang für den jetzt ausgewählten Comic.

Die Ausgabe dieses Programms sieht wie folgt aus:

```
Downloading page http://xkcd.com...
Downloading image http://imgs.xkcd.com/comics/phone_alarm.png...
Downloading page http://xkcd.com/1358/...
Downloading image http://imgs.xkcd.com/comics/nro.png...
```

```
Downloading page http://xkcd.com/1357/...
Downloading image http://imgs.xkcd.com/comics/free_speech.png...
Downloading page http://xkcd.com/1356/...
Downloading image http://imgs.xkcd.com/comics/orbital_mechanics.png...
Downloading page http://xkcd.com/1355/...
Downloading image http://imgs.xkcd.com/comics/airplane_message.png...
Downloading page http://xkcd.com/1354/...
Downloading image http://imgs.xkcd.com/comics/heartbleed_explanation.png...
-- schnipp --
```

Dieses Projekt ist ein gutes Beispiel für ein Programm, das die Verfolgung von Links zum Abruf großer Datenmengen aus dem Web automatisiert. Was Beautiful Soup sonst noch zu bieten hat, können Sie der Dokumentation auf <http://www.crummy.com/software/BeautifulSoup/bs4/doc/> entnehmen.

### Vorschläge für ähnliche Programme

Seiten herunterladen und Links verfolgen sind Grundaufgaben vieler Web-Scraping-Programme. Solche Programme können auch Folgendes tun:

- Allen internen Links auf einer Website folgen, um sie komplett zu sichern
- Alle Posts in einem Webforum kopieren
- Den Katalog der Artikel in einem Online-Store kopieren

Solange Sie den URL herausfinden können, den Sie an `requests.get()` übergeben müssen, sind `requests` und Beautiful Soup eine großartige Hilfe. Manchmal ist dieser URL jedoch schwer zu finden. Es kann auch sein, dass Sie sich auf der betreffenden Website erst anmelden müssen. Mit dem Modul `selenium` können Sie in Ihren Programmen aber auch solche anspruchsvollen Aufgaben ausführen.

### Den Browser mit dem Modul Selenium steuern

Mit dem Modul Selenium kann ein Python-Programm den Browser direkt steuern, indem es Links folgt und Anmeldeinformationen eingibt, als ob ein Mensch mit der Zielseite umgehen würde. Außerdem macht dieses Modul eine weit anspruchsvollere Interaktion mit Webseiten möglich, als `requests` und Beautiful Soup es erlauben. Da es einen Webbrowser startet, ist es allerdings etwas langsamer und schwer im Hintergrund auszuführen, wenn Sie etwa nur einige Dateien aus dem Web herunterladen wollen.

Eine ausführlichere Anleitung zur Installation von Drittanbietermodulen erhalten Sie in Anhang A.

## Einen seleniumgesteuerten Browser starten

Für diese Beispiele brauchen Sie Firefox. Wenn Sie diesen Browser noch nicht haben, können Sie ihn von <http://getfirefox.com/> herunterladen.

Die Module für Selenium zu importieren, ist nicht ganz einfach. Statt `import selenium` müssen Sie `from selenium import webdriver` ausführen. (Eine Erklärung, warum Selenium auf diese Weise aufgebaut ist, würde den Rahmen dieses Buchs sprengen.) Nachdem Sie das getan haben, können Sie Firefox wie folgt mit Selenium starten:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('http://inventwithpython.com')
```

Wenn Sie `webdriver.Firefox()` aufrufen, wird Firefox gestartet. Der Rückgabewert von `webdriver.Firefox()` ist, wie ein Aufruf von `type()` zeigt, vom Datentyp Web-Driver. Wenn Sie `browser.get('http://inventwithpython.com')` aufrufen, wird der Browser zu dem angegebenen URL geleitet. Wenn Sie das Beispiel ausprobieren, sollte Ihr Browser so aussehen wie in Abb. 11–7.



Abb. 11–7 Nach dem Aufruf von `webdriver.Firefox()` und `get()` wird Firefox gestartet.

## Elemente auf der Seite finden

WebDriver-Objekte verfügen über eine Reihe von Methoden, um Elemente auf einer Seite zu finden. Es gibt dabei zwei grundlegende Arten von Methoden, nämlich `find_element_*` und `find_elements_*`: Während die `find_element_*`-Methoden nur ein einziges WebElement-Objekt für das erste übereinstimmende Element auf der Seite zurückgeben, liefern die `find_elements_*`-Methoden eine Liste von WebElement-Objekten für alle übereinstimmenden Elemente.

Methode	Zurückgegebene WebElement-Objekte bzw. -Listen
<code>browser.find_element_by_class_name(name)</code> <code>browser.find_elements_by_class_name(name)</code>	Elemente mit der angegebenen CSS-Klasse
<code>browser.find_element_by_css_selector(selector)</code> <code>browser.find_elements_by_css_selector(selector)</code>	Elemente mit dem angegebenen CSS-Selektor
<code>browser.find_element_by_id(id)</code> <code>browser.find_elements_by_id(id)</code>	Elemente mit dem angegebenen Wert für das Attribut id
<code>browser.find_element_by_link_text(text)</code> <code>browser.find_elements_by_link_text(text)</code>	<a>-Elemente mit genau dem angegebenen Linktext
<code>browser.find_element_by_partial_link_text(text)</code> <code>browser.find_elements_by_partial_link_text(text)</code>	<a>-Elemente, deren Link den angegebenen Text enthält
<code>browser.find_element_by_name(name)</code> <code>browser.find_elements_by_name(name)</code>	Elemente mit dem angegebenen Namen
<code>browser.find_element_by_tag_name(name)</code> <code>browser.find_elements_by_tag_name(name)</code>	Elemente mit dem angegebenen Tag (wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird; d. h. ein <a>-Element wird sowohl bei der Suche nach 'a' als auch nach 'A' gefunden)

**Tab. 11–3** WebDriver-Methoden von Selenium für die Suche nach Elementen

Außer bei den `*_by_tag_name()`-Methoden wird bei den Argumenten zwischen Groß- und Kleinschreibung unterschieden. Wird auf der Seite kein übereinstimmendes Element gefunden, löst das Selenium-Modul die Ausnahme `NoSuchElement` aus. Wenn Sie nicht wollen, dass Ihr Programm in einem solchen Fall abstürzt, fügen Sie entsprechende `try`- und `except`-Anweisungen hinzu.

Wenn Sie über ein `WebElement`-Objekt verfügen, können Sie mehr darüber herausfinden, indem Sie seine Attribute lesen oder seine Methoden aufrufen. Beide sind in Tabelle 11–3 aufgeführt.

Attribut oder Methode	Beschreibung
<code>tag_name</code>	Der Name des Tags, z. B. 'a' für ein <a>-Element
<code>get_attribute(name)</code>	Der Wert des Attributs mit dem angegebenen Namen
<code>text</code>	Der Text innerhalb des Elements, z. B. 'hello' in <span>hello</span>
<code>clear()</code>	Löscht den eingegebenen Text in ein- und mehrzeiligen Texteingabefeldern
<code>is_displayed()</code>	Gibt True zurück, wenn das Element sichtbar ist, anderenfalls False
<code>is_enabled()</code>	Gibt bei Eingabeelementen True zurück, wenn sie aktiviert sind, anderenfalls False

Attribut oder Methode	Beschreibung
is_selected()	Gibt bei Kontrollkästchen und Optionsfeldern True zurück, wenn das Element ausgewählt ist, anderenfalls False
location	Ein Dictionary mit den Schlüsseln 'x' und 'y' für die Position des Elements auf der Seite

**Tab. 11-4** Attribute und Methoden von WebElement-Objekten

Geben Sie zum Ausprobieren das folgende Programm in den Dateieditor ein:

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Found <%s> element with that class name!' % (elem.tag_name))
except:
    print('Was not able to find an element with that name.')
```

Dieses Programm startet Firefox und öffnet darin einen URL. Auf der aufgerufenen Seite versuchen wir, Elemente mit dem Klassennamen 'bookcover' zu finden. Wenn ein solches Element vorhanden ist, geben wir mithilfe des Attributs `tag_name` seinen Tagnamen aus, anderenfalls eine entsprechende Meldung.

Die Ausgabe sieht wie folgt aus:

```
Found <img> element with that class name!
```

Das bedeutet, dass ein `<img>`-Tag mit dem Klassennamen 'bookcover' gefunden wurde.

## Auf Links klicken

Die von den `find_element_*`- und `find_elements_*`-Methoden zurückgegebenen WebElement-Objekte verfügen über die Methode `click()`, die einen Mausklick auf das Element simuliert. Damit können Sie einem Link folgen, einen Optionsschalter auswählen, eine *Submit*-Schaltfläche betätigen oder irgendeinen anderen Vorgang auslösen, der beim Klick auf das Element abläuft. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://inventwithpython.com')
```

```
>>> linkElem = browser.find_element_by_link_text('Read It Online')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click() # follows the "Read It Online" link
```

Dadurch wird <http://inventwithpython.com/> in Firefox geöffnet, das WebElement-Objekt für das <a>-Element mit dem Text *Read It Online* abgerufen und dann ein Klick auf dieses Element simuliert. Der Browser folgt nun diesem Link, als hätten Sie selbst darauf geklickt.

## Formulare ausfüllen und absenden

Um einen simulieren Tastendruck zu einem Textfeld auf einer Webseite zu senden, müssen Sie das <input>- bzw. <textarea>-Element dieses Felds ermitteln und dann die Methode `send_keys()` dafür aufrufen, wie das folgende Beispiel zeigt:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://gmail.com')
>>> emailElem = browser.find_element_by_id('Email')
>>> emailElem.send_keys('not_my_real_email@gmail.com')
>>> passwordElem = browser.find_element_by_id('Passwd')
>>> passwordElem.send_keys('12345')
>>> passwordElem.submit()
```

Sofern Gmail die IDs der Textfelder für Benutzername und Passwort seit der Herausgabe dieses Buchs nicht geändert hat, füllt der vorstehende Code diese Felder mit dem angegebenen Text aus. (Sie können die `id`-Attribute auch mit den Entwicklertools im Browser überprüfen.) Wenn Sie die Methode `submit()` für ein Element aufrufen, hat das die gleiche Wirkung, als ob Sie auf die *Submit*-Schaltfläche des Formulars klicken, in dem sich das Element befindet. (Das Ergebnis wäre das gleiche, wenn Sie im vorstehenden Code einfach `emailElem.submit()` aufrufen.)

## Die Betätigung von Sondertasten simulieren

Selenium verfügt auch über ein Modul für Sondertasten, die sich nicht als Stringwerte eingeben lassen. Die Werte für diese Tasten sind in den Attributen des Moduls `selenium.webdriver.common.keys` gespeichert. Da dieser Modulname ziemlich lang ist, sollten Sie am Anfang des Programms `from selenium.webdriver.common.keys import Keys` ausführen, sodass Sie statt dieser langen Bezeichnung einfach `Keys` schreiben können. Die gängigen `Keys`-Variablen finden Sie in Tabelle 11–5.

Attribut	Bedeutung
Keys.DOWN, Keys.UP, Keys.LEFT, Keys.RIGHT	Die Pfeiltasten
Keys.ENTER, Keys.RETURN	Die Tasten <code>Enter</code> und <code>Return</code>
Keys.HOME, Keys.END, Keys.PAGE_DOWN, Keys.PAGE_UP	Die Tasten <code>Pos1</code> , <code>Ende</code> , <code>Bild abwärts</code> und <code>Bild aufwärts</code>
Keys.ESCAPE, Keys.BACK_SPACE, Keys.DELETE	Die Tasten <code>Esc</code> , <code>Rückschritt</code> und <code>Entf</code>
Keys.F1, Keys.F2 ... Keys.F12	Die Funktionstasten <code>F1</code> bis <code>F12</code>
Keys.TAB	Die Taste <code>Tab</code>

**Tab. 11-5** Häufig verwendete Variablen im Modul `selenium.webdriver.common.keys`

Wenn sich der Cursor nicht in einem Textfeld befindet, können Sie ihn beispielsweise mit den Tasten `Pos1` und `Ende` zum oberen bzw. unteren Rand des Bildschirms versetzen. Geben Sie folgenden Code in die interaktive Shell ein und beobachten Sie, wie die Aufrufe von `send_keys()` durch die Seite scrollen:

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('http://nostarch.com')
>>> htmlElem = browser.find_element_by_tag_name('html')
>>> htmlElem.send_keys(Keys.END) # Scrollt nach unten
>>> htmlElem.send_keys(Keys.HOME) # Scrollt nach oben
```

`<html>` ist das Grundtag in allen HTML-Dateien: Der gesamte Inhalt der HTML-Datei befindet sich zwischen den Tags `<html>` und `</html>`. mit dem Aufruf `browser.find_element_by_tag_name('html')` senden Sie daher simulierte Tastenbetätigungen an die Webseite im Ganzen. Das kann nützlich sein, wenn beispielsweise beim Scrollen zum unteren Rand der Seite neue Inhalte geladen werden.

## Auf Browserschaltflächen klicken

Mit den folgenden Methoden kann Selenium auch Klicks auf Browserschaltflächen simulieren:

- `browser.back()` Klickt auf die Schaltfläche *Zurück*.
- `browser.forward()` Klickt auf die Schaltfläche *Weiter*.
- `browser.refresh()` Klickt auf die Schaltfläche *Aktualisieren* (bzw. *Seite neu laden*).
- `browser.quit()` Klickt auf die Windows-Schaltfläche *Schließen*.

## Weitere Informationen über Selenium

Selenium bietet noch viel mehr als nur die hier beschriebenen Funktionen. Sie können damit die Cookies im Browser ändern, einen Screenshot einer Webseite aufnehmen oder eigenen JavaScript-Code ausführen. Mehr darüber erfahren Sie in der Selenium-Dokumentation auf <http://selenium-python.readthedocs.org/>.

## Zusammenfassung

Bei den meisten langweiligen Aufgaben geht es nicht nur um Dateien auf Ihrem Computer. Wenn Sie programmgesteuert Webseiten herunterladen können, ist das eine nützliche Erweiterung Ihrer Programme. Das Modul `requests` vereinfacht diesen Vorgang, mit einigen Grundkenntnissen in HTML und Selektoren können Sie das Modul BeautifulSoup nutzen, um die heruntergeladenen Seiten zu durchsuchen.

Um webgestützte Aufgaben komplett zu automatisieren, brauchen Sie jedoch die vollständige Kontrolle über den Browser, die Ihnen das Modul Selenium gewährt. Damit können Sie sich automatisch auf Websites anmelden und Formulare ausfüllen. Da ein Webbrowser das übliche Instrument darstellt, um Informationen über das Internet zu senden und zu empfangen, ist dieses Modul ein wichtiger Bestandteil Ihres Programmierer-Werkzeugkastens.

## Wiederholungsfragen

1. Beschreiben Sie kurz die Unterschiede zwischen den Modulen `webbrowser`, `requests`, Beautiful Soup und Selenium.
2. Welche Art von Objekt gibt `requests.get()` zurück? Wie können Sie auf den heruntergeladenen Inhalt als Stringwert zugreifen?
3. Welche `requests`-Methode prüft, ob der Download funktioniert hat?
4. Wie können Sie den HTTP-Statuscode einer `requests`-Antwort abrufen?
5. Wie speichern Sie eine `requests`-Antwort in einer Datei?
6. Welches Tastaturkürzel verwenden Sie, um die Entwickertools im Browser zu öffnen?
7. Wie können Sie (in den Entwickertools) den HTML-Code eines bestimmten Elements auf einer Webseite anzeigen?
8. Welchen CSS-Selektorstring verwenden Sie, um ein Element mit dem `id`-Attribut `main` zu finden?
9. Welchen CSS-Selektorstring verwenden Sie, um Elemente mit der CSS-Klasse `highlight` zu finden?
10. Welchen CSS-Selektorstring verwenden Sie, um alle `<div>`-Elemente zu finden, die in einem anderen `<div>`-Element verschachtelt sind?

11. Welchen CSS-Selektorstring verwenden Sie, um das <button>-Element mit dem value-Attribut favorite zu finden?
12. Nehmen Sie an, in der Variablen `spam` ist das Tag-Objekt von Beautiful Soup für das Element <div>Hello world!</div> gespeichert. Wie rufen Sie den String 'Hello world!' aus diesem Objekt ab?
13. Wie speichern Sie alle Attribute eines Tag-Objekts von Beautiful Soup in der Variablen `linkElem`?
14. Der Befehl `import selenium` funktioniert nicht. Wie importieren Sie das Modul Selenium auf korrekte Weise?
15. Was ist der Unterschied zwischen den `find_element_*`- und den `find_elements_*`-Methoden?
16. Welche `WebElement`-Methoden bietet Selenium zum Simulieren von Mausklicks und Tastenbetätigungen?
17. Welche einfachere Möglichkeit bietet Selenium zum Absenden eines Formulars, anstatt `send_keys(Keys.ENTER)` für das `WebElement`-Objekt der *Submit*-Schaltfläche aufzurufen?
18. Wie können Sie mit Selenium einen Klick auf die Browserschaltflächen *Weiter*, *Zurück* und *Aktualisieren* simulieren?

## **Übungsprojekte**

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erfüllen:

### **E-Mail-Programm für die Befehlszeile**

Schreiben Sie ein Programm, das an der Befehlszeile eine E-Mail-Adresse und einen Textstring entgegennimmt und sich dann mithilfe von Selenium an Ihrem E-Mail-Konto anmeldet und eine E-Mail mit dem String als Text an die angegebene Adresse sendet. (Für dieses Programm sollten Sie ein eigenes E-Mail-Konto einrichten.) Damit können Sie anderen Programmen eine Benachrichtigungsfunktion hinzufügen. Sie können auch ähnliche Programme schreiben, die Nachrichten von einem Facebook- oder Twitter-Konto senden.

### **Download-Programm für Fotowebseiten**

Schreiben Sie ein Programm, das eine Fotowebseite wie Flickr oder Imgur aufsucht, nach einer bestimmten Kategorie von Fotos sucht und alle resultierenden Bilder herunterlädt. Es ist sogar möglich, ein Programm zu schreiben, das auf allen Foto-websites mit Suchfunktion funktioniert.

## 2048

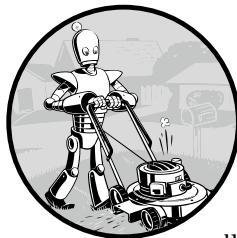
2048 ist ein einfaches Spiel, bei dem Sie Kacheln anordnen, indem Sie sie mit den Pfeiltasten nach oben, nach unten, nach links oder nach rechts verschieben. Dabei können Sie einen ziemlich hohen Punktestand erreichen, indem Sie das Muster »nach oben, nach rechts, nach unten, nach links« ständig wiederholen. Schreiben Sie ein Programm, das das Spiel auf <https://gabrielecirulli.github.io/2048/> öffnet und dann wiederholt die Pfeiltasten nach oben, nach rechts, nach unten und nach links betätigt, um das Spiel automatisch zu spielen.

## Linküberprüfung

Schreiben Sie ein Programm, das den URL einer Webseite entgegennimmt und dann versucht, jede auf dieser Seite verlinkte Seite herunterzuladen. Das Programm sollte alle Seiten markieren, für die der Statuscode 404 (»Not found«) zurückgegeben wird, und sie als defekte Links ausgeben.

# 12

## Arbeiten mit Excel-Arbeitsblättern



Excel ist eine weit verbreitete und leistungsfähige Tabellenkalkulationsanwendung für Windows. Das Modul `openpyxl` macht es möglich, in Python-Programmen Excel-Arbeitsblatdateien zu lesen und zu bearbeiten. Damit können Sie sich beispielsweise der langweiligen Aufgabe entledigen, bestimmte Daten von einem Arbeitsblatt in ein anderes zu kopieren oder Tausende von Zeilen zu durchsuchen, um nur an einigen wenigen davon, die bestimmte Kriterien erfüllen, winzige Änderungen vorzunehmen.

Oder stellen Sie sich vor, Sie müssten Hunderte von Arbeitsblättern mit Abteilungsbudgets nach denjenigen durchforsten, die im Minus sind. Dies alles sind genau die Arten von langweilen Arbeiten mit Arbeitsblättern, die Python Ihnen abnehmen kann.

Excel ist eine proprietäre Software von Microsoft, doch gibt es auch freie Alternativen für Windows, OS X und Linux. Sowohl LibreOffice Calc als auch OpenOffice Calc können mit dem von Excel verwendeten Dateiformat `.xlsx` für Arbeitsmappen umgehen. Das bedeutet, dass Sie das Modul `openpyxl` auch für Arbeitsblätter dieser Anwendungen einsetzen können. Die Software können Sie von <https://www.libreoffice.org/> bzw. <http://www.openoffice.org/> herunterladen.

Schauen Sie sich diese Programme ruhig auch an, wenn Sie Excel schon installiert haben, denn vielleicht stellen Sie ja fest, dass Sie damit besser zureckkommen. Die Screenshots in diesem Kapitel stammen jedoch alle von Excel 2010 in Windows 7.

## Excel-Dokumente

Als Erstes müssen wir einige Definitionen festlegen: Ein Excel-Dokument wird als *Arbeitsmappe* bezeichnet und in einer Datei mit der Erweiterung *.xlsx* gespeichert. Jede Arbeitsmappe wiederum kann aus mehreren *Arbeitsblättern* bestehen. Das Arbeitsblatt, das der Benutzer gerade betrachtet (oder vor dem Schließen von Excel zuletzt betrachtet hat), ist das *aktive Arbeitsblatt*.

Jedes Arbeitsblatt besteht aus *Spalten* (bezeichnet mit Buchstaben, beginnend mit A) und *Zeilen* (nummeriert, beginnend mit 1). Das Feld im Schnittpunkt einer Spalte und einer Zeile wird als *Zelle* bezeichnet. Jede Zelle kann eine Zahl oder einen Textwert enthalten. Das Raster der Zellen mit Daten bildet das Arbeitsblatt.

## Das Modul openpyxl installieren

OpenPyXL ist im Lieferumfang von Python nicht enthalten, sodass Sie es erst installieren müssen. Folgen Sie dazu den Anweisungen zur Installation von Drittanbietermodulen aus Anhang A. Der Name des Moduls lautet `openpyxl`. Um zu prüfen, ob es korrekt installiert wurde, geben Sie in der interaktiven Shell Folgendes ein:

```
>>> import openpyxl
```

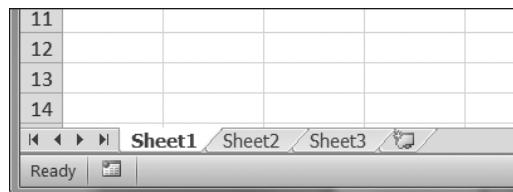
Wenn Sie keine Fehlermeldung erhalten, wurde das Modul ordnungsgemäß installiert. Denken Sie daran, das Modul `openpyxl` zu importieren, bevor Sie die Beispiele in diesem Kapitel ausführen, da Sie sonst die Fehlermeldung `NameError: name 'openpyxl' is not defined` erhalten.

In diesem Buch behandeln wir die Version 2.1.4 von OpenPyXL, allerdings werden regelmäßig neue Versionen veröffentlicht. Machen Sie sich darüber aber keine Sorgen, denn die neuen Versionen sollten noch längere Zeit rückwärtskompatibel mit den hier vorgestellten Anweisungen sein. Wenn Sie eine neuere Version haben und mehr über die zusätzlichen Merkmale erfahren wollen, die sie bietet, schauen Sie sich die vollständige Dokumentation auf <http://openpyxl.readthedocs.org/> an.

## Excel-Dokumente lesen

Für die Beispiele in diesem Kapitel verwenden wir die Arbeitsmappe *example.xlsx* im aktuellen Arbeitsverzeichnis. Sie können dieses Dokument entweder

selbst erstellen oder von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen. Abb. 12–1 zeigt die Registerkarten der drei Standardarbeitsblätter *Sheet1*, *Sheet2* und *Sheet3* (bzw. *Tabelle1* usw. in der deutschen Version), die Excel in neuen Arbeitsmappen automatisch anlegt. (Die Anzahl der Standardarbeitsmappen kann je nach Betriebssystem und Tabellenkalkulationsprogramm unterschiedlich sein.)



**Abb. 12–1** Registerkarten für die Arbeitsblätter einer Arbeitsmappe in der linken unteren Ecke von Excel

Das erste Arbeitsblatt in dieser Beispieldatei sollte so aussehen wie in Tabelle 12–1. (Wenn Sie die Datei *example.xlsx* nicht von der Website herunterladen, geben Sie diese Daten ein, um die Datei selbst zu erstellen.)

	A	B	C
1	4/5/2015 1:34:02 PM	Apples	73
2	4/5/2015 3:41:23 AM	Cherries	85
3	4/6/2015 12:46:51 PM	Pears	14
4	4/8/2015 8:59:43 AM	Oranges	52
5	4/10/2015 2:07:00 AM	Apples	152
6	4/10/2015 6:10:37 PM	Bananas	23
7	4/10/2015 2:40:46 AM	Strawberries	98

**Tab. 12–1** Die Arbeitsmappe *example.xlsx*

Damit haben wir nun eine Beispiel-Arbeitsmappe, die wir mit dem Modul `openpyxl` bearbeiten können.

### Excel-Dokumente mit OpenPyXL öffnen

Wenn Sie das Modul `openpyxl` importiert haben, können Sie die Funktion `openpyxl.load_workbook()` verwenden. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

Diese Funktion nimmt einen Dateinamen entgegen und gibt einen Wert vom Datentyp `workbook` (= Arbeitsmappe) zurück. Dieses `Workbook`-Objekt steht für eine Excel-Datei, ebenso wie ein `File`-Objekt für eine geöffnete Textdatei steht.

Denken Sie daran, dass sich `example.xlsx` im aktuellen Arbeitsverzeichnis befinden muss, damit Sie damit arbeiten können. Um das Arbeitsverzeichnis zu ermitteln, importieren Sie `os` und rufen `os.getcwd()` auf. Mit `os.chdir()` können Sie das Arbeitsverzeichnis auch ändern.

## Arbeitsblätter aus der Arbeitsmappe abrufen

Um eine Liste mit den Namen der Arbeitsblätter in der Arbeitsmappe abzurufen, verwenden Sie die Methode `get_sheet_names()`:

```
>>> import openpyxl  
>>> wb = openpyxl.load_workbook('example.xlsx')  
>>> wb.get_sheet_names()  
['Sheet1', 'Sheet2', 'Sheet3']  
>>> sheet = wb.get_sheet_by_name('Sheet3')  
>>> sheet  
<Worksheet "Sheet3">  
>>> type(sheet)  
<class 'openpyxl.worksheet.worksheet.Worksheet'>  
>>> sheet.title  
'Sheet3'  
>>> anotherSheet = wb.get_active_sheet()  
>>> anotherSheet  
<Worksheet "Sheet1">
```

Für ein Arbeitsblatt gibt es ein `Worksheet`-Objekt, das Sie abrufen können, indem Sie den Namen des Arbeitsblatts an die `Workbook`-Methode `get_sheet_by_name()` übergeben. Mit der Methode `get_active_sheet()` des `Workbook`-Objekts können Sie außerdem das aktive Arbeitsblatt abrufen, also dasjenige, das in Excel geöffnet ist. Wenn Sie ein `Worksheet`-Objekt haben, können Sie den Namen des Arbeitsblatts aus dem Attribut `title` gewinnen.

## Zellen in Arbeitsblättern ansprechen

Wenn Sie über ein `Worksheet`-Objekt verfügen, können Sie auf die `Cell`-Objekte in diesem Arbeitsblatt anhand von deren Namen zugreifen:

```
>>> import openpyxl  
>>> wb = openpyxl.load_workbook('example.xlsx')  
>>> sheet = wb.get_sheet_by_name('Sheet1')  
>>> sheet['A1']  
<Cell Sheet1.A1>
```

```
>>> sheet['A1'].value
datetime.datetime(2015, 4, 5, 13, 34, 2)
>>> c = sheet['B1']
>>> c.value
'Apples'
>>> 'Row ' + str(c.row) + ', Column ' + c.column + ' is ' + c.value
'Row 1, Column B is Apples'
>>> 'Cell ' + c.coordinate + ' is ' + c.value
'Cell B1 is Apples'
>>> sheet['C1'].value
73
```

Das Attribut `value` eines `Cell`-Objekts enthält, wie der Name schon sagt, den Wert, der in der betreffenden Zelle gespeichert ist. Des Weiteren verfügen `Cell`-Objekte über die Attribute `row`, `column` und `coordinate`, die Informationen über die Position der Zelle enthalten.

Hier liefert der Zugriff auf das `value`-Attribut des `Cell`-Objekts für Zelle B1 den String 'Apples'. Das Attribut `row` enthält den Integer 1, das Attribut `column` den String 'B' und das Attribut `coordinates` den Wert 'B1'.

OpenPyXL interpretiert die Daten in Spalte A automatisch und gibt sie als `datetime`-Werte statt als Strings zurück. Der Datentyp `datetime` wird in Kapitel 16 genauer erklärt.

Eine Spalte anhand ihres Kennbuchstabens anzusprechen, kann bei der Programmierung ziemlich knifflig sein, vor allem da die Spalten hinter Spalte Z zweistellige Bezeichnungen wie AA, AB usw. tragen. Alternativ können Sie die Zelle auch mit der Methode `cell()` des Arbeitsblatts abrufen und für beide Schlüsselwortargumente `row` und `column` Integerwerte übergeben. Der Integer für die erste Zeile bzw. Spalte ist dabei jeweils 1 und nicht 0. Führen Sie damit das Beispiel in der interaktiven Shell wie folgt fort:

```
>>> sheet.cell(row=1, column=2)
<Cell Sheet1.B1>
>>> sheet.cell(row=1, column=2).value
'Apples'
>>> for i in range(1, 8, 2):
    print(i, sheet.cell(row=i, column=2).value)

1 Apples
3 Pears
5 Apples
7 Strawberries
```

Hier übergeben Sie der Methode `cell()` des Arbeitsblatts `row=1` und `column=2` und erhalten damit das `Cell`-Objekt für die Zelle B1 zurück, als ob Sie `sheet['B1']` angegeben hätten. Anschließend durchlaufen Sie mit der Methode `cell()` und

ihren Schlüsselwortargumenten eine for-Schleife, um die Werte einer Folge von Zellen auszugeben.

Nehmen wir an, Sie wollen die Werte in allen Zellen der Spalte B mit ungerader Zeilennummer ausgeben. Wenn Sie als Schrittweitenparameter der Funktion range() die Zahl 2 übergeben, werden die Zellen jeder zweiten Zeile abgerufen (hier die von allen ungeradzahligen Zeilen). Die Variable i der for-Schleife wird als Schlüsselwortargument row an cell() übergeben, während das Schlüsselwortargument column stets 2 ist. Beachten Sie, dass hier der Integer 2 übergeben wird, nicht der String 'B'.

Die Größe des Arbeitsblatts können Sie mit den Worksheet-Methoden get\_highest\_row() und get\_highest\_column() ermitteln:

```
>>> import openpyxl  
>>> wb = openpyxl.load_workbook('example.xlsx')  
>>> sheet = wb.get_sheet_by_name('Sheet1')  
>>> sheet.get_highest_row()  
7  
>>> sheet.get_highest_column()  
3
```

Beachten Sie, dass get\_highest\_column() nicht den Buchstaben zurückgibt, der in Excel erscheint, sondern einen Integer.

## Umrechnen zwischen Kennbuchstaben und Nummern

Um aus den Kennbuchstaben Spaltennummern zu machen, rufen Sie die Funktion openpyxl.cell.column\_index\_from\_string() auf. Wollen Sie umgekehrt aus diesen Zahlen den Spaltenbuchstaben ermitteln, verwenden Sie openpyxl.cell.get\_column\_letter():

```
>>> import openpyxl  
>>> from openpyxl.cell import get_column_letter, column_index_from_string  
>>> get_column_letter(1)  
'A'  
>>> get_column_letter(2)  
'B'  
>>> get_column_letter(27)  
'AA'  
>>> get_column_letter(900)  
'AHP'  
>>> wb = openpyxl.load_workbook('example.xlsx')  
>>> sheet = wb.get_sheet_by_name('Sheet1')  
>>> get_column_letter(sheet.get_highest_column())  
'C'
```

```
>>> column_index_from_string('A')
1
>>> column_index_from_string('AA')
27
```

Nachdem Sie diese Funktionen aus dem Modul `openpyxl.cell` importiert haben, können Sie `get_column_letter()` aufrufen und einen Integer wie 27 übergeben, um herauszufinden, welchen Buchstaben die 27. Spalte aufweist. Die Funktion `column_index_string()` macht das Gegenteil: Sie nimmt einen Spaltenbuchstaben entgegen und teilt Ihnen mit, welche laufende Nummer die Spalte hat. Um diese Funktionen zu nutzen, muss keine Arbeitsmappe geladen sein. Wenn Sie wollen, können Sie aber eine Arbeitsmappe laden, ein `Worksheet`-Objekt abrufen, dessen Methode `get_highest_column()` aufrufen, um den Integer für die letzte Spalte zu ermitteln, und diesen Wert dann an `get_column_letter()` übergeben.

## Zeilen und Spalten eines Arbeitsblatts abrufen

Sie können einen Ausschnitt eines `Worksheet`-Objekts bilden, um alle `Cell`-Objekte in einer Zeile, einer Spalte oder einem rechteckigen Teilbereich abzurufen. Dann können Sie alle Zellen in diesem Ausschnitt in einer Schleife durchlaufen. Probieren Sie das folgende Beispiel in der interaktiven Shell aus:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> tuple(sheet['A1':'C3'])
((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>), (<Cell Sheet1.A2>,
<Cell Sheet1.B2>, <Cell Sheet1.C2>), (<Cell Sheet1.A3>, <Cell Sheet1.B3>,
<Cell Sheet1.C3>))
>>> for rowOfCellObjects in sheet['A1':'C3']:    ❶
    for cellObj in rowOfCellObjects:    ❷
        print(cellObj.coordinate, cellObj.value)
        print('--- END OF ROW ---')
A1 2015-04-05 13:34:02
B1 Apples
C1 73
--- END OF ROW ---
A2 2015-04-05 03:41:23
B2 Cherries
C2 85
--- END OF ROW ---
A3 2015-04-06 12:46:51
B3 Pears
C3 14
--- END OF ROW ---
```

Mit diesem Code geben wir an, dass wir alle Cell-Objekte in dem rechteckigen Bereich von A1 bis C3 haben wollen. Zurückgegeben wird ein Generator-Objekt mit allen Zellen in diesem Bereich. Um dieses Objekt grafisch übersichtlicher darzustellen, verwenden wir `tuple()`, um die enthaltenen Cell-Objekte in einem Tupel darzustellen.

Dieses Tupel wiederum enthält drei weitere Tupel, nämlich eines für jede Zeile, von der obersten Zeile des gewünschten Bereichs bis zur untersten. Diese drei inneren Tupel wiederum enthalten die Cell-Objekte einer Zeile des gewünschten Bereichs, von der linken bis zur rechten Zelle. Damit umfasst unser Ausschnitt des Arbeitsblatts alle Cell-Objekte des Bereichs von A1 bis C3, von der Zelle in der oberen linken Ecke bis zu der in der unteren rechten Ecke.

Um die Werte all dieser Zellen auszugeben, verwenden wir zwei for-Schleifen. Die äußere durchläuft alle Zeilen in dem Ausschnitt (❶) und für jede Zeile durchläuft die verschachtelte Schleife dann alle darin enthaltenen Zellen (❷).

Um auf die Werte in den Zellen einer bestimmten Zeile oder Spalte zuzugreifen, können Sie auch die Attribute `rows` und `columns` des Worksheet-Objekts verwenden:

```
>>> import openpyxl  
>>> wb = openpyxl.load_workbook('example.xlsx')  
>>> sheet = wb.get_active_sheet()  
>>> sheet.columns[1]  
(<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>,  
<Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)  
>>> for cell0bj in sheet.columns[1]:  
    print(cell0bj.value)  
  
Apples  
Cherries  
Pears  
Oranges  
Apples  
Bananas  
Strawberries
```

Aus dem `rows`-Attribut eines Worksheet-Objekts gewinnen Sie ein Tupel aus Tupeln. Jedes der inneren Tupel steht für eine Zeile und enthält die darin enthaltenen Cell-Objekte. Auch das Attribut `columns` liefert ein Tupel aus Tupeln, bei dessen inneren Tupeln es sich um die Cell-Objekte in einer einzelnen Spalte handelt. Da es in `example.xlsx` sieben Zeilen und drei Spalten gibt, liefert `rows` ein Tupel aus sieben Tupeln (mit je drei Cell-Objekten) und `columns` ein Tupel aus drei Tupeln (mit je sieben Cell-Objekten).

Um auf ein bestimmtes Tupel zuzugreifen, geben wir seinen Index im umfassenden Tupel an, beispielsweise `sheets.columns[1]` für das Tupel, das für Spalte B steht. Für das Tupel mit den `Cell`-Objekten in Spalte A müssen wir dagegen `sheets.columns[0]` verwenden. Wenn Sie ein Tupel für eine Zeile oder Spalte haben, können Sie die darin enthaltenen `Cell`-Objekte in einer Schleife durchlaufen und deren Werte ausgeben.

## Arbeitsmappen, Arbeitsblätter und Zellen

Zur Übersicht sehen Sie hier noch eine Zusammenfassung der Vorgehensweise, um Zellen aus einer Arbeitsmappendatei abzurufen:

1. Importieren Sie das Modul `openpyxl`.
2. Rufen Sie die Funktion `openpyxl.load_workbook()` auf.
3. Rufen Sie ein `Workbook`-Objekt ab.
4. Rufen Sie die `Workbook`-Methode `get_active_sheet()` oder `get_sheet_by_name()` auf.
5. Rufen Sie ein `Worksheet`-Objekt ab.
6. Verwenden Sie die Indizes oder die `Worksheet`-Methode `cell()` mit den Schlüsselwortargumenten `row` und `column`.
7. Rufen Sie ein `Cell`-Objekt ab.
8. Lesen Sie den Wert des `value`-Attributs für das `Cell`-Objekt.

## Projekt: Daten in einer Arbeitsmappe lesen

Stellen Sie sich vor, Sie haben die langweilige Aufgabe, Tausende von Zeilen in einer Arbeitsmappe mit den Daten der US-Volkszählung 2010 durchzugeben, um die Bevölkerungszahl und die Anzahl der Erhebungsgebiete in jedem Landkreis zu ermitteln. (Ein Erhebungsgebiet, in der Arbeitsmappe als *Census Tract* bezeichnet, ist lediglich ein zu Erhebungszwecken festgelegter geografischer Raum.) Jede Zeile steht dabei für ein Erhebungsgebiet. Die Datei mit dieser Arbeitsmappe trägt den Namen `censuspopdata.xlsx` und kann von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) heruntergeladen werden. Der Inhalt sieht wie in Abb. 12–2 aus.

1	A	B	C	D	E
	CensusTract	State	County	POP2010	
9841	06075010500	CA	San Francisco	2685	
9842	06075010600	CA	San Francisco	3894	
9843	06075010700	CA	San Francisco	5592	
9844	06075010800	CA	San Francisco	4578	
9845	06075010900	CA	San Francisco	4320	
9846	06075011000	CA	San Francisco	4827	
9847	06075011100	CA	San Francisco	5164	
<b>Population by Census Tract</b>					
Ready					

Abb. 12–2 Die Excel-Datei *censuspopdata.xlsx*

Excel kann zwar die Summe des Inhalts mehrerer markierter Zellen berechnen, doch dazu müssen Sie trotzdem noch die Zellen für jeden der mehr als 3000 Landkreise markieren. Selbst wenn Sie die Bevölkerungszahl eines Landkreises manuell in wenigen Sekunden ermitteln können, würde es Stunden dauern, die gesamte Arbeitsmappe durchzugehen.

In diesem Projekt schreiben Sie ein Skript, das die Daten aus dieser Volkszählungsdatei liest und dann in wenigen Sekunden die Statistiken für die einzelnen Landkreise berechnet.

Das Programm führt folgende Aufgaben aus:

- Daten aus einer Excel-Arbeitsmappe lesen
- Anzahl der Erhebungsgebiete in jedem Landkreis zählen
- Gesamtbevölkerung in jedem Landkreis ermitteln
- Ergebnisse ausgeben

Dazu muss der Code Folgendes tun:

- Die Zellen eines Excel-Dokuments mithilfe des Moduls `openpyxl` öffnen und lesen
- Die gewünschten Daten über Erhebungsgebiete und Bevölkerungszahlen berechnen und in einer Datenstruktur speichern
- Die Datenstruktur mithilfe des Moduls `pprint` in einer Textdatei mit der Endung `.py` speichern

### Schritt 1: Die Daten der Arbeitsmappe lesen

In der Arbeitsmappe *censuspopdata.xlsx* gibt es nur ein Arbeitsblatt, nämlich *Population by Census Tract*. Jede Zeile enthält die Daten für ein einziges Erhebungsgebiet. Die Spalten geben die Nummer des Erhebungsgebiets (A), die Abkürzung

für den Bundesstaat (B), den Namen des Landkreises (C) und die Bevölkerung in dem Erhebungsgebiet (D) an.

Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *readCensusExcel.py*.

```
#! python3
# readCensusExcel.py - Berechnet Bevölkerung und Anzahl der Erhebungsgebiete
# für jeden Landkreis

import openpyxl, pprint ❶
print('Opening workbook...')
wb = openpyxl.load_workbook('censuspopdata.xlsx') ❷
sheet = wb.get_sheet_by_name('Population by Census Tract') ❸
countyData = {}

# TODO: countyData mit Bevölkerung und Anzahl der Erhebungsgebiete des
# Landkreises füllen
print('Reading rows...')
for row in range(2, sheet.get_highest_row() + 1): ❹
    # Jede Zeile im Arbeitsblatt enthält die Daten eines Erhebungsgebiets
    state = sheet['B' + str(row)].value
    county = sheet['C' + str(row)].value
    pop = sheet['D' + str(row)].value

# TODO: Neue Textdatei öffnen und Inhalt von countyData hineinschreiben
```

Dieser Code importiert zunächst sowohl `openpyxl` als auch das Modul `pprint`, mit dem Sie später die fertigen Landkreisdaten ausgeben werden (❶). Anschließend öffnet er die Datei *censuspopdata.xlsx* (❷) und ruft das Arbeitsblatt mit den Volkszählungsdaten ab (❸), um dort die einzelnen Zeilen zu durchlaufen (❹).

Des Weiteren wird hier die Variable `countyData` erstellt, die die für jeden einzelnen Landkreis ermittelten Daten über die Bevölkerung und die Anzahl der Landkreise aufnimmt. Bevor Sie irgendetwas darin speichern können, müssen Sie jedoch die genaue Struktur der darin aufzubewahrenden Daten ermitteln.

## Schritt 2: Die Datenstruktur füllen

Die Datenstruktur in `countyData` ist ein Dictionary mit den Abkürzungen der Bundesstaaten als Schlüssel. Die Werte zu diesen Schlüsseln sind wiederum Dictionaries, als deren Schlüssel Strings mit den Namen der Landkreise in dem entsprechenden Staat fungieren. Der Wert für jeden Landkreis ist wiederum ein Dictionary, das aber nur zwei Schlüssel hat, nämlich `'tracts'` und `'pop'`. Deren Werte sind die Anzahl der Erhebungsgebiete (`tracts`) und die Bevölkerungszahl (`population`). Das Gesamt-Dictionary sieht also wie folgt aus:

```
{'AK': {'Aleutians East': {'pop': 3141, 'tracts': 1},
        'Aleutians West': {'pop': 5561, 'tracts': 2},
        'Anchorage': {'pop': 291826, 'tracts': 55},
        'Bethel': {'pop': 17013, 'tracts': 3},
        'Bristol Bay': {'pop': 997, 'tracts': 1},
-- schnipp --
```

Wenn wir dieses Dictionary in countyData speichern, können wir die Bevölkerung und die Anzahl von Erhebungsgebieten mit folgenden Ausdrücken daraus abrufen:

```
>>> countyData['AK']['Anchorage']['pop']
291826
>>> countyData['AK']['Anchorage']['tracts']
55
```

Allgemein gesagt sehen die Schlüssel des Dictionarys countyData wie folgt aus:

```
countyData[Abk. Bundesstaat][Landkreis]['tracts']
countyData[Abk. Bundesstaat][Landkreis]['pop']
```

Da Sie nun die Struktur von countyData kennen, können Sie Code schreiben, um das Dictionary mit den Daten der Landkreise zu füllen. Ergänzen Sie das Programm wie folgt:

```
#! python3
# readCensusExcel.py - Berechnet Bevölkerung und Anzahl der Erhebungsgebiete
# für jeden Landkreis

-- schnipp --

for row in range(2, sheet.get_highest_row() + 1):    ❸
    # Jede Zeile im Arbeitsblatt enthält die Daten eines Erhebungsgebiets
    state = sheet['B' + str(row)].value
    county = sheet['C' + str(row)].value
    pop = sheet['D' + str(row)].value

    # Stellt sicher, dass der Schlüssel für den Staat vorhanden ist
    countyData.setdefault(state, {})    ❹
    # Stellt sicher, dass der Schlüssel für den Landkreis in dem Staat
    # vorhanden ist
    countyData[state].setdefault(county, {'tracts': 0, 'pop': 0})    ❺❻

    # Jede Zeile steht für ein Erhebungsgebiet, daher wird die Summe um 1
    # erhöht
    countyData[state][county]['tracts'] += 1    ❼
    # Erhöht die Summe um die Bevölkerungszahl im Erhebungsgebiet
    countyData[state][county]['pop'] += int(pop)    ❽

# TODO: Neue Textdatei öffnen und Inhalt von countyData hineinschreiben
```

Die letzten beiden neuen Codezeilen führen die eigentliche Berechnungsarbeit durch, nämlich die Inkrementierung des Werts für tracts (❸) und die Erhöhung des Werts von pop (❹) für den aktuellen Landkreis in jeder Iteration der for-Schleife.

Der restliche Code ist erforderlich, da Sie nicht einfach ein Landkreis-Dictionary als Wert für einen Bundesstaatschlüssel hinzufügen können, wenn es diesen Schlüssel in countyData noch gar nicht gibt. (Beispielsweise ruft countyData['AK'] ['Anchorage'] ['tracts'] += 1 einen Fehler hervor, wenn der Schlüssel 'AK' noch nicht vorhanden ist.) Um sicherzustellen, dass der Schlüssel für den Bundesstaat in der Datenstruktur existiert, müssen Sie die Methode setdefault() aufrufen, um einen Wert für den Staat festzulegen, falls noch keiner vorhanden ist (❺).

Ebenso wie das Dictionary countyData ein Dictionary als Wert für jeden Bundesstaatenschlüssel hat, so haben *diese* Dictionarys ein Dictionary als Wert für jeden einzelnen Landkreisschlüssel (❻) und diese Dictionarys wiederum brauchen die Schlüssel 'tract' und 'pop', die zu Anfang den Integerwert 0 haben. (Wenn Sie bei dieser Dictionary-Struktur jemals den Faden verlieren, schauen Sie sich das Beispiel-Dictionary am Anfang dieses Abschnitts an.)

Da setdefault() nichts tut, wenn der Schlüssel bereits vorhanden ist, können Sie diese Funktion bei jeder Iteration der for-Schleife problemlos ausführen.

### Schritt 3: Die Ergebnisse in eine Datei schreiben

Nach dem Abschluss der for-Schleife enthält das Dictionary countyData alle Informationen über die Bevölkerung und die Anzahl der Erhebungsgebiete nach Landkreisen und Bundesstaaten geordnet. Jetzt können Sie weiteren Code hinzufügen, um diese Ergebnisse in eine Textdatei oder eine andere Excel-Arbeitsmappe zu schreiben. Hier wollen wir jedoch die Funktion pprint.pformat() verwenden, um das Dictionary countyData als langen String in die Datei *census2010.py* zu schreiben. Ergänzen Sie am Ende des Programms folgenden Code (achten Sie darauf, ihn nicht einzurücken, damit er außerhalb der for-Schleife steht):

```
#! python3
# readCensusExcel.py - Berechnet Bevölkerung und Anzahl der Erhebungsgebiete
# für jeden Landkreis

-- schnipp --

for row in range(2, sheet.get_highest_row() + 1):

-- schnipp --
```

```
# Öffnet eine neue Textdatei und schreibt den Inhalt von countyData hinein
print('Writing results...')
resultFile = open('census2010.py', 'w')
resultFile.write('allData = ' + pprint.pformat(countyData))
resultFile.close()
print('Done.')
```

Die Funktion `pprint.pformat()` erzeugt einen String, der als gültiger Python-Code formatiert ist. Dadurch, dass Sie ihn in die Textdatei `census2010.py` ausgeben, haben Sie aus Ihrem Python-Programm heraus ein Python-Programm generiert. Das mag sich übermäßig kompliziert anhören, doch der Vorteil besteht darin, dass Sie `census2010.py` nun wie jedes andere Python-Modul importieren können. Ändern Sie das aktuelle Arbeitsverzeichnis in der interaktiven Shell auf den Ordner mit der neuen Datei `census2010.py` (auf meinem Laptop ist das C:\Python34) und importieren Sie diese:

```
>>> import os
>>> os.chdir('C:\\\\Python34')
>>> import census2010
>>> census2010.allData['AK']['Anchorage']
{'pop': 291826, 'tracts': 55}
>>> anchoragePop = census2010.allData['AK']['Anchorage']['pop']
>>> print('The 2010 population of Anchorage was ' + str(anchoragePop))
The 2010 population of Anchorage was 291826
```

Das Programm `readCensusExcel.py` ist Wegwerfcode: Nachdem Sie die Ergebnisse in der Datei `census2010.py` gespeichert haben, werden Sie das Programm nicht noch einmal ausführen. Wenn Sie die Landkreisdaten benötigen, führen Sie einfach `import census2010` aus.

Die manuelle Berechnung dieser Daten hätte Sie Stunden gekostet; dieses Programm erledigt die gleiche Aufgabe in Sekunden. Mit OpenPyXL haben Sie nun keine Schwierigkeiten mehr, um Informationen aus Excel-Arbeitsmappen zu gewinnen und Berechnungen damit anzustellen. Das komplette Programm können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.

## Vorschläge für ähnliche Programme

In vielen Unternehmen und Büros wird Excel verwendet, um verschiedene Arten von Daten zu speichern. Es ist dabei nicht unüblich, dass die Arbeitsblätter groß und unhandlich werden. Alle Programme, die Excel-Dokumente durchforsten, weisen einen ähnlichen Aufbau auf: Sie laden die Arbeitsmappendatei, bereiten einige Variablen oder Datenstrukturen vor und durchlaufen dann alle Zeilen in der Arbeitsmappe. Mit einem solchen Programm können Sie u. A. folgende Aufgaben ausführen:

- Daten in verschiedenen Zeilen einer Arbeitsmappe vergleichen
- Mehrere Excel-Dateien öffnen und Daten in verschiedenen Arbeitsmappen vergleichen
- Eine Arbeitsmappe auf leere Zellen oder ungültige Inhalte in Zellen überprüfen und die Benutzer gegebenenfalls darauf hinweisen
- Daten aus einer Arbeitsmappe lesen und als Eingabe für Python-Programme nutzen

## Excel-Dokumente schreiben

OpenPyXL bietet ebenso eine Möglichkeit, um Daten zu schreiben, sodass Sie in Ihren Python-Programmen auf einfache Weise auch Arbeitsmappen mit Tausenden von Datenzeilen erstellen und bearbeiten können.

### Excel-Dokumente erstellen und speichern

Mit der Funktion `openpyxl.Workbook` erstellen Sie ein neues, leeres Workbook-Objekt:

```
>>> import openpyxl  
>>> wb = openpyxl.Workbook()  
>>> wb.get_sheet_names()  
['Sheet']  
>>> sheet = wb.get_active_sheet()  
>>> sheet.title  
'Sheet'  
>>> sheet.title = 'Spam Bacon Eggs Sheet'  
>>> wb.get_sheet_names()  
['Spam Bacon Eggs Sheet']
```

Die Arbeitsmappe hat zu Anfang nur ein einziges Arbeitsblatt namens *Sheet*. Um dessen Namen zu ändern, speichern Sie den gewünschten String im Attribut `title`.

Wenn Sie ein Workbook-Objekt, seine Arbeitsblätter oder Zellen bearbeiten, wird die Datei erst gespeichert, wenn Sie die Workbook-Methode `save()` aufrufen. Das können Sie wie folgt in der interaktiven Shell ausprobieren (wobei sich *example.xlsx* im aktuellen Arbeitsverzeichnis befinden muss):

```
>>> import openpyxl  
>>> wb = openpyxl.load_workbook('example.xlsx')  
>>> sheet = wb.get_active_sheet()  
>>> sheet.title = 'Spam Spam Spam'  
>>> wb.save('example_copy.xlsx')
```

Um die Änderung des Arbeitsblattnamens zu speichern, müssen wir einen Dateinamen als String an `save()` übergeben. Wenn wir dabei einen anderen als den ursprünglichen Dateinamen angeben, wie hier `example_copy.xlsx` statt `example.xlsx`, wird eine Kopie der Arbeitsmappe gespeichert.

Wenn Sie eine Arbeitsmappe aus einer Datei laden und bearbeiten, sollten Sie sie anschließend unter einem anderen Dateinamen speichern, damit Sie die ursprüngliche Version immer noch zur Verfügung haben, falls ein Bug im Code dazu geführt hat, dass in der neuen Datei falsche oder beschädigte Daten stehen.

## Arbeitsblätter erstellen und entfernen

Mit den Methoden `create_sheet()` und `remove_sheet()` können Sie Arbeitsblätter zu einer Arbeitsmappe hinzufügen bzw. daraus entfernen:

```
>>> import openpyxl  
>>> wb = openpyxl.Workbook()  
>>> wb.get_sheet_names()  
['Sheet']  
>>> wb.create_sheet()  
<Worksheet "Sheet1">  
>>> wb.get_sheet_names()  
['Sheet', 'Sheet1']  
>>> wb.create_sheet(index=0, title='First Sheet')  
<Worksheet "First Sheet">  
>>> wb.get_sheet_names()  
['First Sheet', 'Sheet', 'Sheet1']  
>>> wb.create_sheet(index=2, title='Middle Sheet')  
<Worksheet "Middle Sheet">  
>>> wb.get_sheet_names()  
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

Die Methode `create_sheet()` gibt ein neues `Worksheet`-Objekt namens `SheetX` zurück, das standardmäßig als letztes Arbeitsblatt in der Arbeitsmappe eingerichtet ist. Optional können Sie Index und Name des neuen Arbeitsblatts auch in den Schlüsselwortargumenten `index` und `title` übergeben.

Setzen Sie das vorstehende Beispiel wie folgt fort:

```
>>> wb.get_sheet_names()  
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']  
>>> wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet'))  
>>> wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))  
>>> wb.get_sheet_names()  
['First Sheet', 'Sheet']
```

Die Methode `remove_sheet()` nimmt als Argument ein `Worksheet`-Objekt entgegen, nicht den String des Arbeitsblattnamens. Wenn Sie nur den Namen des zu entfernenden Blatts kennen, rufen Sie `get_sheet_by_name()` auf und übergeben Sie den Rückgabewert an `remove_sheet()`.

Nachdem Sie Arbeitsblätter hinzugefügt oder entfernt haben, müssen Sie `save()` aufrufen, um die Änderungen zu speichern.

## Werte in Zellen schreiben

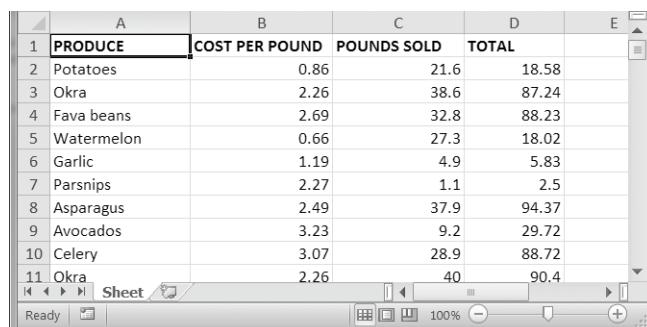
Werte können Sie auf ähnliche Weise in Zellen schreiben wie in ein Dictionary:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
>>> sheet['A1'] = 'Hello world!'
>>> sheet['A1'].value
'Hello world!'
```

Um anzugeben, in welche Zelle Sie Werte schreiben wollen, übergeben Sie dem `Worksheet`-Objekt einen String mit den Koordinaten der gewünschten Zelle – vergleichbar mit dem Schlüssel eines Dictionarys.

## Projekt: Ein Arbeitsblatt aktualisieren

In diesem Projekt schreiben Sie ein Programm, um die Zellen in einem Arbeitsblatt mit Daten über Obst- und Gemüseverkäufe zu aktualisieren. Das Programm durchsucht das Arbeitsblatt, sucht nach gegebenen Sorten und aktualisiert deren Preis. Laden Sie die Arbeitsmappe von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter. Einen Ausschnitt davon sehen Sie in Abb. 12–3.



The screenshot shows a Microsoft Excel spreadsheet with data in columns A through E. Column A contains product names, column B contains the cost per pound, column C contains the pounds sold, and column D contains the total cost. Row 1 is a header row with the labels: PRODUCE, COST PER POUND, POUNDS SOLD, and TOTAL. The data rows are numbered 2 to 11. The last row (row 11) has a different background color. The bottom of the screen shows the standard Excel ribbon tabs (File, Home, Insert, etc.) and a status bar indicating 'Ready' and 'Sheet 1'.

	A	B	C	D	E
1	PRODUCE	COST PER POUND	POUNDS SOLD	TOTAL	
2	Potatoes	0.86	21.6	18.58	
3	Okra	2.26	38.6	87.24	
4	Fava beans	2.69	32.8	88.23	
5	Watermelon	0.66	27.3	18.02	
6	Garlic	1.19	4.9	5.83	
7	Parsnips	2.27	1.1	2.5	
8	Asparagus	2.49	37.9	94.37	
9	Avocados	3.23	9.2	29.72	
10	Celery	3.07	28.9	88.72	
11	Okra	2.26	40	90.4	

Abb. 12–3 Ein Arbeitsblatt mit Zahlen zu Obst- und Gemüseverkäufen

Jede Zeile steht für einen einzelnen Verkaufsvorgang. Die Spalten geben die Art der verkauften Früchte oder des Gemüses an (A), die Kosten pro Pfund dieses Erzeugnisses (B), die verkaufte Menge in Pfund (C) und die Einnahmen für den Verkauf (D). Die Spalte *TOTAL* enthält die Excel-Formel  $=ROUND(B3 * C3, 2)$ , die die Kosten pro Pfund mit der Menge in Pfund multipliziert und das Ergebnis auf den nächstliegenden Centbetrag runden. Wenn es in den Spalten B oder C irgendwelche Änderungen gibt, wird der Wert in der zugehörigen Zelle der Spalte *TOTAL* unmittelbar aufgrund dieser Formel aktualisiert.

Nehmen wir aber an, jemand hat versehentlich falsche Preise für Knoblauch, Staudensellerie und Zitronen eingegeben, sodass Sie nun die langweilige Aufgabe haben, Tausende von Zeilen in diesem Arbeitsblatt durchzugehen, um die Kosten pro Pfund in allen Zeilen zu ändern, in denen es um diese Erzeugnisse geht. Sie können das nicht einfach mit Suchen und Ersetzen erledigen, da es auch andere Artikel mit demselben Pfundpreis geben kann, die Sie nicht versehentlich »korrigieren« wollen. Bei Tausenden von Zeilen kann es Stunden dauern, diese Aufgabe manuell zu erledigen. Sie können aber auch ein kleines Programm schreiben, das sie in wenigen Sekunden erledigt.

Dieses Programm muss Folgendes tun:

- Alle Zeilen durchlaufen
- In allen Zeilen für Knoblauch, Staudensellerie und Zitronen den Preis ändern

Der Code muss also folgende Aufgaben erfüllen:

- Die Datei mit dem Arbeitsblatt öffnen
- In jeder Spalte prüfen, ob der Wert in Spalte A Celery, Garlic oder Lemon lautet
- Wenn ja, den Preis in Spalte B ändern
- Das Arbeitsblatt in einer neuen Datei speichern (damit Sie die alte Version für alle Fälle aufbewahren)

### Schritt 1: Eine Datenstruktur mit den neuen Informationen einrichten

Die neuen Preise für die drei Erzeugnisse lauten:

Staudensellerie (Celery)	1,19
Knoblauch (Garlic)	3,07
Zitronen (Lemon)	1,27

Theoretisch könnten Sie folgenden Code schreiben:

```
if produceName == 'Celery':  
    cellObj = 1.19  
if produceName == 'Garlic':  
    cellObj = 3.07  
if produceName == 'Lemon':  
    cellObj = 1.27
```

Es wäre allerdings ziemlich unelegant, die Produktnamen und Preise auf diese Weise hartkodiert in das Programm zu schreiben. Wenn Sie das Arbeitsblatt später für andere Produkte oder mit anderen Preisen noch einmal aktualisieren wollen, müssten Sie eine Menge Code ändern und bei jeder Codeänderung besteht die Gefahr, dass sich Bugs einschleichen.

Eine flexiblere Lösung besteht darin, die Preisinformationen in einem Dictionary abzulegen und im Code auf diese Datenstruktur zurückzugreifen. Geben Sie im Dateieditor folgenden Code ein:

```
#! python3  
# updateProduce.py - Korrigiert Preise in einem Arbeitsblatt über Obst- und  
# Gemüseverkäufe  
  
import openpyxl  
  
wb = openpyxl.load_workbook('produceSales.xlsx')  
sheet = wb.get_sheet_by_name('Sheet')  
  
# Die Erzeugnisse und ihre neuen Preise  
PRICE_UPDATES = {'Garlic': 3.07,  
                  'Celery': 1.19,  
                  'Lemon': 1.27}  
  
# TODO: Die Zeilen durchlaufen und die Preise aktualisieren
```

Speichern Sie dieses Programm als *updateProduce.py*. Wenn Sie das Arbeitsblatt später erneut aktualisieren müssen, brauchen Sie nur das Dictionary `PRICE_UPDATES` zu ändern und keinen anderen Code.

## Schritt 2: Alle Zeilen prüfen und die falschen Preise korrigieren

Der nächste Teil des Programms durchläuft alle Zeilen in dem Arbeitsblatt. Fügen Sie am Ende von *updateProduce.py* folgenden Code hinzu:

```
#! python3  
# updateProduce.py - Korrigiert Preise in einem Arbeitsblatt über Obst- und  
# Gemüseverkäufe  
  
-- schnipp --
```

```

# Durchläuft die Zeilen und aktualisiert die Preise
for rowNum in range(2, sheet.get_highest_row()): # Überspringt 1. Zeile ❶
    produceName = sheet.cell(row=rowNum, column=1).value ❷
    if produceName in PRICE_UPDATES: ❸
        sheet.cell(row=rowNum, column=2).value = PRICE_UPDATES[produceName]

wb.save('updatedProductSales.xlsx') ❹

```

Da Zeile 1 den Tabellenkopf enthält, durchlaufen wir die Zeilen beginnend mit Zeile 2 (❶). Die Zelle in Spalte 1 (d. h. Spalte A) wird in der Variablen `produceName` gespeichert (❷). Wenn der Wert von `produceName` als Schlüssel im Dictionary `PRICE_UPDATES` vorhanden ist (❸), dann handelt es sich um eine Zeile, deren Preis korrigiert werden muss. Der richtige Preis ist in `PRICE_UPDATES[produceName]` zu finden.

Beachten Sie, wie sauber der Code dank der Verwendung von `PRICE_UPDATES` wird: Wir brauchen hier nur eine einzige `if`-Anweisung, anstatt für jedes einzelne zu ändernde Erzeugnis Code wie `if produceName == 'Garlic':` zu schreiben. Da der Code das Dictionary `PRICE_UPDATES` nutzt, anstatt die Produktnamen und Preise in einer `for`-Schleife hinzukodieren, müssen Sie nur dieses Dictionary ändern, wenn Sie weitere Änderungen an dem Arbeitsblatt vornehmen wollen, und keinen weiteren Code schreiben.

Nachdem er das gesamte Arbeitsblatt durchlaufen und die Änderungen vorgenommen hat, speichert der Code das Workbook-Objekt in `updatedProductSales.xlsx` (❹). Die alte Version wird also nicht überschrieben, sondern steht für den Fall zur Verfügung, dass es in dem Programm einen Bug gibt und das neue Arbeitsblatt falsch ist. Nachdem Sie sich vergewissert haben, dass das aktualisierte Arbeitsblatt richtig aussieht, können Sie das alte löschen.

Den kompletten Quellcode für dieses Programm können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.

### Vorschläge für ähnliche Programme

Da bei der Büroarbeit sehr oft Excel-Arbeitsblätter verwendet werden, ist ein Programm, das Excel-Dateien automatisch bearbeiten und schreiben kann, wirklich nützlich. Mit einem solchen Programm können Sie folgende Aufgaben ausführen:

- Daten in einer Arbeitsmappe lesen und Teile daraus in eine andere Arbeitsmappe schreiben
- Daten von Websites, aus Textdateien oder der Zwischenablage lesen und in ein Arbeitsblatt schreiben
- Daten in Arbeitsblättern automatisch bereinigen. Beispielsweise kann ein solches Programm mithilfe eines regulären Ausdrucks verschiedene Formate von Telefonnummern erkennen und sie zu einem Standardformat vereinheitlichen.

## Die Schrift in den Zellen gestalten

Durch die Gestaltung der Schrift in einzelnen Zellen, Zeilen oder Spalten können Sie wichtige Bereiche in einem Arbeitsblatt hervorheben. Beispielsweise können wir in unserem Obst- und Gemüsearbeitsblatt die Zeilen für Kartoffeln, Knoblauch und Pastinaken durch Fettschrift hervorheben oder alle Zeilen mit einem Pfundpreis von mehr als 5 \$ kursiv stellen. Eine manuelle Gestaltung von ausgesuchten Bereichen des Arbeitsblatts wäre ziemlich mühselig, doch ein Programm kann dies im Nu erledigen.

Um die Schriftart in den Zellen anpassen zu können, müssen Sie die Funktionen `Font()` und `Style()` des Moduls `openpyxl.styles` importieren.

```
from openpyxl.styles import Font, Style
```

Dadurch können Sie einfach `Font()` statt `openpyxl.styles.Font()` eingeben. (Mehr über diese Form der `import`-Anweisung erfahren Sie im Abschnitt »Module importieren« in Kapitel 2.)

Der folgende Beispielcode erstellt eine neue Arbeitsmappe und setzt Zelle A1 auf eine 24-Punkt-Kursivschrift:

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
>>> italic24Font = Font(size=24, italic=True) ❶
>>> styleObj = Style(font=italic24Font) ❷
>>> sheet['A'].style = styleObj ❸
>>> sheet['A1'] = 'Hello world!'
>>> wb.save('styled.xlsx')
```

Die gesammelten Stileinstellungen für eine Zelle werden in OpenPyXL durch ein `Style`-Objekt ausgedrückt, das im Attribut `style` des `Cell`-Objekts gespeichert ist. Um den Stil einer Zelle festzulegen, weisen Sie ihrem `style`-Attribut ein `Style`-Objekt zu.

In diesem Beispiel gibt `Font(size=24, italic=True)` ein `Font`-Objekt zurück, das in `italic24font` gespeichert wird (❶). Die Schlüsselwortargumente `size` und `italic` von `Font()` legen die `style`-Attribute des `Font`-Objekts fest. Anschließend wird dieses `Font`-Objekt im Aufruf von `Style(font=italic24Font)` übergeben. Diese Funktion gibt den Wert zurück, den Sie in `styleObj` gespeichert haben (❷). Wenn Sie `styleObj` dem `style`-Attribut der Zelle A1 zuweisen (❸), werden alle darin enthaltenen Informationen über die Schriftgestaltung auf sie angewandt.

## Font-Objekte

Die style-Attribute in Font-Objekten bestimmen, wie der Text in den Zellen angezeigt wird. Um die Attribute zur Schriftgestaltung festzulegen, übergeben Sie der Funktion `Font()` entsprechende Schlüsselwortargumente. Eine Liste dieser Argumente finden Sie in Tabelle 12–2.

Schlüsselwortargument	Datentyp	Beschreibung
<code>name</code>	String	Der Name der Schriftart, z. B. 'Calibri' oder 'Times New Roman'
<code>size</code>	Integer	Die Schriftgröße in Punkt
<code>bold</code>	<code>bold</code>	True für Fettschrift
<code>italic</code>	<code>Boolean</code>	True für Kursivschrift

**Tab. 12–2** Schlüsselwortargumente für style-Attribute von Font-Objekten

Rufen Sie `Font()` auf, um ein Font-Objekt zu erstellen, und speichern Sie dieses Font-Objekt dann in einer Variablen. Anschließend übergeben Sie diese Variable an `Style()` und speichern das resultierende Style-Objekt in einer weiteren Variablen, die Sie wiederum dem style-Attribut eines `Cell`-Objekts zuweisen. Der folgende Beispielcode erstellt auf diese Weise verschiedene Schriftstile:

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')

>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> styleObj1 = Style(font=fontObj1)
>>> sheet['A1'].style=styleObj1
>>> sheet['A1'] = 'Bold Times New Roman'

>>> fontObj2 = Font(size=24, italic=True)
>>> styleObj2 = Style(font=fontObj2)
>>> sheet['B3'].style=styleObj2
>>> sheet['B3'] = '24 pt Italic'

>>> wb.save('styles.xlsx')
```

Hier speichern wir ein Font-Objekt in `fontObj1` und verwenden es, um ein Style-Objekt zu erstellen, das wir in `styleObj1` ablegen. Anschließend setzen wir das style-Attribut des `Cell`-Objekts für Zelle A1 auf `styleObj1`. Diesen Vorgang wiederholen wir mit einem weiteren Font- und einem weiteren Style-Objekt, um den

Stil der zweiten Zelle festzulegen. Wenn Sie diesen Code ausführen, weisen die Zellen A1 und B3 des Arbeitsblatts einen eigenen Schriftstil auf, wie Sie in Abb. 12–4 sehen.

	A	B	C	D
1	<b>Times New Roman</b>			
2				
3		<i>24 pt Italic</i>		
4				
5				

**Abb. 12–4** Ein Arbeitsblatt mit eigener Schriftgestaltung

Für Zelle A1 haben wir als Schriftnamen 'Times New Roman' verwendet und **bold** auf True gesetzt, damit der Text in einer fetten Times New Roman erscheint. Da wir keine Größe angegeben haben, wird der OpenPyXL-Standard von 11 Punkt verwendet. Der Text in Zelle B3 dagegen ist kursiv und weist eine Größe von 24 Punkt auf. Hier ist es die Schriftart, die wir nicht angegeben haben, weshalb die Standardschrift von OpenPyXL verwendet wird, nämlich Calibri.

## Formeln

Mithilfe von Formeln, die stets mit einem Gleichheitszeichen beginnen, ist es möglich, dass Zellen Werte enthalten, die aus den Werten in anderen Zellen berechnet wurden. In diesem Abschnitt nutzen wir openpyxl, um programmgesteuert Formeln in Zellen einzufügen. Das geht genauso wie das Einfügen regulärer Werte:

```
>>> sheet['B9'] = '=SUM(B1:B8)'
```

Damit wird `=SUM(B1:B8)` als Wert in Zelle B9 gespeichert. Diese Formel berechnet die Summe der Werte in den Zellen B1 bis B8 und zeigt sie in Zelle B9 an, wie Sie in Abb. 12–5 sehen:

The screenshot shows a Microsoft Excel spreadsheet. Row 1 contains the value 82 in cell B2. Rows 2 through 8 each contain a value: 11, 85, 18, 57, 51, 38, and 42 respectively. Row 9 is labeled "TOTAL:" and contains the formula =SUM(B1:B8) in cell B9, which has a black border. The result of the sum, 384, is displayed in cell C9. The spreadsheet has three tabs at the bottom: Sheet1, Sheet2, and Sheet3. The status bar at the bottom left says "Ready".

	A	B	C	D	E
1		82			
2		11			
3		85			
4		18			
5		57			
6		51			
7		38			
8		42			
9	TOTAL:	384			
10					

Abb. 12–5 Zelle B9 enthält die Formel =SUM(B1:B8), die den Inhalt der Zellen B1 bis B8 summiert

Eine Formel wird wie jeder andere Textwert in einer Zelle festgelegt:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet['A1'] = 200
>>> sheet['A2'] = 300
>>> sheet['A3'] = '=SUM(A1:A2)'
>>> wb.save('writeFormula.xlsx')
```

Die Zellen A1 und A2 werden auf die Werte 200 bzw. 300 gesetzt, A3 dagegen auf eine Formel, die den Inhalt von A1 und A2 addiert. Wenn Sie dieses Arbeitsblatt in Excel öffnen, zeigt A3 den Wert 500 an.

Die Formeln in den Zellen können ebenso gelesen werden wie alle anderen Werte. Wenn Sie aber statt an der Formel als solcher an dem *Ergebnis* der Berechnung interessiert sind, müssen Sie True als Schlüsselwortargument `data_only` an `load_workbook()` übergeben. Ein Workbook-Objekt kann daher entweder nur die Formeln oder die Ergebnisse der Formeln anzeigen, aber nicht beides. (Allerdings können Sie für ein und dieselbe Arbeitsmappendatei mehrere Workbook-Objekte laden.) Um sich den Unterschied zwischen dem Laden einer Arbeitsmappe mit und ohne das Schlüsselwortargument `data_only` anzusehen, geben Sie Folgendes in die interaktive Shell ein:

```
>>> import openpyxl
>>> wbFormulas = openpyxl.load_workbook('writeFormula.xlsx')
>>> sheet = wbFormulas.get_active_sheet()
>>> sheet['A3'].value
'=SUM(A1:A2)'
```

```
>>> wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx', data_only=True)
>>> sheet = wbDataOnly.get_active_sheet()
>>> sheet['A3'].value
500
```

Wenn `load_workbook()` mit `data_only=True` geladen wird, zeigt Zelle A3 den Wert 500 an, also das Ergebnis der Formel `=SUM(A1:A2)` statt des Textes der Formel.

Excel-Formeln bieten gewisse Programmiermöglichkeiten für Arbeitsblätter, doch bei komplizierteren Aufgaben können sie ziemlich schnell unhandlich werden. Selbst wenn Sie eingehend mit Excel-Formeln vertraut sind, ist es geradezu ein Albtraum, herauszufinden, was `=IFERROR(TRIM(IF(LEN(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE))>0, SUBSTITUTE(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE), " ", ""),"")),"")` bewirkt. Python-Code ist viel übersichtlicher.

## Das Erscheinungsbild von Zeilen und Spalten festlegen

Die Größe von Zeilen und Spalten können Sie in Excel ganz einfach anpassen, indem Sie auf den Zeilen- oder Spaltenkopf klicken und daran ziehen. Wollen Sie aber die Größe einer Zeile oder Spalte an den Inhalt der Zellen anpassen oder die Größen in sehr vielen Excel-Dateien anpassen, dann geht es schneller, dafür ein Python-Programm zu schreiben.

Es ist auch möglich, Zeilen und Spalten auszublenden. Eine weitere Möglichkeit besteht darin, sie zu fixieren, sodass sie auch beim Scrollen stets angezeigt werden und in einem Ausdruck auf jeder Seite erscheinen (was insbesondere praktisch für Spalten- und Zeilenüberschriften ist).

### Zeilenhöhe und Spaltenbreite festlegen

Worksheet-Objekte verfügen über die Attribute `row_dimensions` und `column_dimensions`, die die Zeilenhöhe bzw. Spaltenbreite festlegen:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet['A1'] = 'Tall row'
>>> sheet['B2'] = 'Wide column'
>>> sheet.row_dimensions[1].height = 70
>>> sheet.column_dimensions['B'].width = 20
>>> wb.save('dimensions.xlsx')
```

Bei beiden Attributen handelt es sich um Dictionary-ähnliche Werte: `row_dimensions` enthält RowDimension-Objekte, `column_dimensions` dagegen ColumnDimension-Objekte. Um auf diese Objekte zuzugreifen, geben Sie in `row_dimensions` die Zeilenummer

an (in unserem Fall 1 oder 2) bzw. in `column_dimensions` den Spaltenbuchstaben (hier A oder B).

Das Arbeitsblatt aus `dimensions.xlsx` sehen Sie in Abb. 12–6.

	A	B	
1	Tall row		
2		Wide column	
3			

**Abb. 12–6** Zeile 1 und Spalte B sind auf eine größere Höhe bzw. Breite eingestellt

Wenn Sie ein `RowDimension`- oder `ColumnDimension`-Objekt abgerufen haben, können Sie die Höhe bzw. Breite dafür festlegen. Die Zeilenhöhe lässt sich auf einen Integer- oder Fließkommawert zwischen 0 und 409 setzen. Diese Werte sind Maßangaben in *Punkt*, wobei ein Punkt 1/72 Zoll ist. Die Standardhöhe einer Zeile beträgt 12,75. Bei der Spaltenbreite sind Integer- und Fließkommawerte von 0 bis 255 möglich. Diese Werte stehen für die Anzahl von Zeichen mit der Standardschriftgröße von 11 Punkt, die in der Zelle dargestellt werden können. Standardmäßig ist eine Spalte 8,43 Zeichen breit. Spalten und Zeilen mit Breiten- bzw. Höhenwerten von 0 sind vor dem Benutzer verborgen.

## Zellen verbinden und aufteilen

Mit der `sheet`-Methode `merge_cells()` können Sie einen rechteckigen Bereich eines Arbeitsblatts zu einer einzigen Zelle verbinden:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet.merge_cells('A1:D3')
>>> sheet['A1'] = 'Twelve cells merged together.'
>>> sheet.merge_cells('C5:D5')
>>> sheet['C5'] = 'Two merged cells.'
>>> wb.save('merged.xlsx')
```

Das Argument von `merge_cells()` ist ein String, der die obere linke und die untere rechte Zelle des zu verbindenden Bereichs angibt. Beispielsweise werden mit '`A1:D3`' zwölf Zellen verschmolzen. Um den Wert in die Gesamtzelle zu schreiben, geben Sie ihn für die oberste linke Zelle der verschmolzenen Gruppe ein.

Wenn Sie den vorstehenden Code ausführen, sieht *merged.xlsx* anschließend wie in Abb. 12–7 aus.

	A	B	C	D	E
1					
2					
3	Twelve cells merged together.				
4					
5		Two merged cells.			
6					
7					

**Abb. 12–7** Verbundene Zellen in einem Arbeitsblatt

Um Zellen wieder aufzuteilen, rufen Sie die Methode `unmerge_cells()` auf:

```
>>> import openpyxl  
>>> wb = openpyxl.load_workbook('merged.xlsx')  
>>> sheet = wb.get_active_sheet()  
>>> sheet.unmerge_cells('A1:D3')  
>>> sheet.unmerge_cells('C5:D5')  
>>> wb.save('merged.xlsx')
```

Wenn Sie die Änderungen speichern und sich das Arbeitsblatt erneut ansehen, werden Sie feststellen, dass die verbundenen Zellen wieder in die einzelnen Zellen zurückverwandelt wurden.

## Bereiche fixieren

Bei Arbeitsblättern, die zu groß sind, um komplett angezeigt werden zu können, ist es hilfreich, einige Zeilen am oberen Rand oder einige Spalten am linken Rand zu fixieren, damit sie immer sichtbar bleiben, auch wenn der Benutzer durch das Arbeitsblatt scrollt. Die Worksheet-Objekte in OpenPyXL verfügen über das Attribut `freeze_panes`, für das Sie ein `Cell`-Objekt oder einen String mit den Koordinaten einer Zelle angeben können. Alle Zeilen oberhalb und alle Spalten links von dieser Zelle werden fixiert, die Zeile und Spalte der angegebenen Zelle selbst aber nicht.

Um die Fixierung aufzuheben, setzen Sie `freeze_panes` auf `None` oder `'A1'`. Tabelle 12–3 zeigt, welche Zeilen und Spalten bei einigen Beispieleinstellungen von `freeze_panes` fixiert werden.

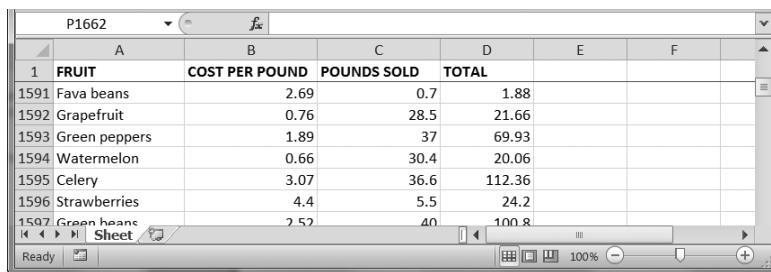
Einstellung von <code>freeze_panes</code>	Fixierte Zeilen und Spalten
<code>sheet.freeze_panes = 'A2'</code>	Zeile 1
<code>sheet.freeze_panes = 'B1'</code>	Spalte A
<code>sheet.freeze_panes = 'C1'</code>	Spalten A und B
<code>sheet.freeze_panes = 'C2'</code>	Zeile 1 und Spalten A und B
<code>sheet.freeze_panes = 'A1' oder sheet.freeze_panes = None</code>	Keine

**Tab. 12–3** Beispiele für fixierte Bereiche

Mit dem Obst- und Gemüsearbeitsblatt von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) im aktuellen Arbeitsverzeichnis geben Sie Folgendes in die interaktive Shell ein:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('produceSales.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.freeze_panes = 'A2'
>>> wb.save('freezeExample.xlsx')
```

Wenn Sie das Attribut `freeze_panes` auf 'A2' setzen, ist Zeile 1 immer zu sehen, unabhängig davon, wie das Arbeitsblatt gescrollt wird. Das können Sie in Abb. 12–8 erkennen.



**Abb. 12–8** Wenn `freeze_panes` auf 'A2' gesetzt ist, wird Zeile 1 immer angezeigt, auch wenn der Benutzer ganz nach unten scrollt.

## Diagramme

OpenPyXL ermöglicht es Ihnen, aus den Daten in den Zellen eines Arbeitsblatts Balken-, Linien-, Streu- und Tortendiagramme zu erstellen. Gehen Sie dazu folgendermaßen vor:

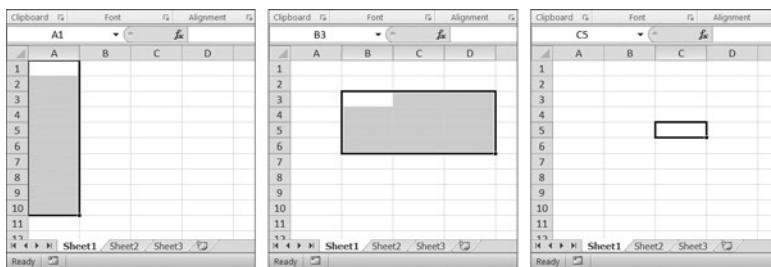
1. Erstellen Sie aus einer rechteckigen Auswahl von Zellen ein Reference-Objekt.
2. Übergeben Sie dieses Reference-Objekt, um ein Series-Objekt zu erstellen.

3. Erstellen Sie ein Chart-Objekt.
4. Hängen Sie das Series-Objekt an das Chart-Objekt an.
5. Legen Sie optional die Werte der Variablen drawing.top, drawing.left, drawing.width und drawing.height des Chart-Objekts fest.
6. Fügen Sie das Chart-Objekt zum Worksheet-Objekt hinzu.

Um ein Reference-Objekt zu erstellen, rufen Sie die Funktion `openpyxl.charts.Reference()` auf und übergeben ihr drei Objekte:

1. Das Worksheet-Objekt, das die Diagrammdaten enthält.
2. Ein Tupel zweier Integer, das für die obere linke Zelle des rechteckigen Bereichs mit den Diagrammdaten steht. Der erste Integer gibt dabei die Zeile an, der zweite die Spalte. Beachten Sie, dass die erste Zeile die Nummer 1 hat, nicht 0.
3. Ein Tupel zweier Integer, das für die untere rechte Zelle des rechteckigen Bereichs mit den Diagrammdaten steht. Der erste Integer gibt dabei die Zeile an, der zweite die Spalte.

Abb. 12–9 zeigt einige Beispielargumente für diese Koordinaten.



**Abb. 12–9** Von links nach rechts: (1, 1), (10, 1); (3, 2), (6, 4); (5, 3), (5, 3)

Geben Sie den folgenden Code in die interaktive Shell ein, um ein Balkendiagramm zu erstellen und zu dem Arbeitsblatt hinzuzufügen:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> for i in range(1, 11): # Fügt einige Daten in Spalte A ein
    sheet['A' + str(i)] = i

>>> ref0bj = openpyxl.charts.Reference(sheet, (1, 1), (10, 1))

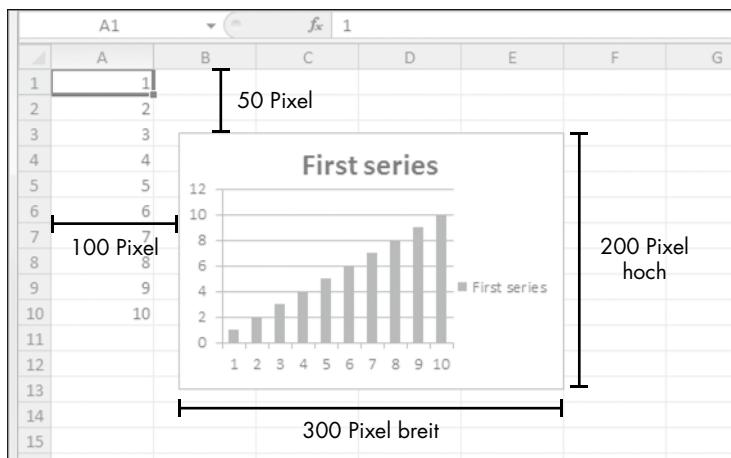
>>> series0bj = openpyxl.charts.Series(ref0bj, title='First series')

>>> chart0bj = openpyxl.charts.BarChart()
>>> chart0bj.append(series0bj)
>>> chart0bj.drawing.top = 50      # Legt die Position fest
```

```
>>> chartObj.drawing.left = 100
>>> chartObj.drawing.width = 300 # Legt die Größe fest
>>> chartObj.drawing.height = 200

>>> sheet.add_chart(chartObj)
>>> wb.save('sampleChart.xlsx')
```

Dadurch ergibt sich das Arbeitsblatt aus Abb. 12–10.



**Abb. 12–10** Ein Arbeitsblatt mit einem Diagramm

Da wir hier `openpyxl.charts.BarChart()` aufgerufen haben, ergibt sich ein Balkendiagramm. Mit den Funktionen `openpyxl.charts.LineChart()`, `openpyxl.charts.ScatterChart()` und `openpyxl.charts.PieChart()` können Sie jedoch auch Linien-, Streu- und Tortendiagramme erstellen.

In der aktuellen Version 2.1.4 von OpenPyXL lädt die Funktion `load_workbook()` leider keine Diagramme in Excel-Dateien: Wenn die Excel-Datei über ein Diagramm verfügt, ist sie in dem geladenen Workbook-Objekt nicht enthalten. Falls Sie daher ein Workbook-Objekt laden und unmittelbar darauf unter demselben Dateinamen speichern, entfernen Sie dadurch jegliche Diagramme aus der Arbeitsmappe.

## Zusammenfassung

Bei der Verarbeitung von Informationen besteht der schwierige Teil meistens gar nicht in der Verarbeitung an sich, sondern darin, an die Daten in dem richtigen Format für das Programm zu kommen. Nachdem Sie eine Arbeitsmappe in Python geladen haben, lassen sich die Daten darin jedoch viel schneller abrufen und bearbeiten als manuell.

Außerdem können Sie Ihre Programme aus Arbeitsmappen ausgeben. Wenn Ihre Kollegen also die Daten mit Tausenden von Kundenkontaktdaten aus einer Text- oder PDF-Datei in Form eines Arbeitsblatts benötigen, dann müssen Sie diese Informationen nicht mühselig von Hand in Excel kopieren.

Mit dem Modul `openpyxl` und einigen Programmierkenntnissen wird sogar die Verarbeitung riesiger Arbeitsblätter zum Kinderspiel.

## Wiederholungsfragen

Nehmen Sie zur Beantwortung der folgenden Fragen an, dass Sie ein Workbook-Objekt in der Variablen `wb` haben, ein Worksheet-Objekt in `sheet`, ein Cell-Objekt in `cell`, ein Comment-Objekt in `comm` und ein Image-Objekt in `img`.

1. Was gibt die Funktion `openpyxl.load_workbook()` zurück?
2. Was gibt die Workbook-Methode `get_sheet_names()` zurück?
3. Wie rufen Sie das Worksheet-Objekt für das Arbeitsblatt 'Sheet1' ab?
4. Wie rufen Sie das Worksheet-Objekt für das aktuelle Arbeitsblatt der Arbeitsmappe ab?
5. Wie rufen Sie den Wert in Zelle C5 ab?
6. Wie setzen Sie den Wert in Zelle C5 auf "Hello"?
7. Wie rufen Sie Zeile und Spalte einer Zelle als Integerwerte ab?
8. Was geben die Arbeitsblattmethoden `get_highest_column()` und `get_highest_row()` zurück? Welchen Datentyp haben diese Rückgabewerte?
9. Welche Funktion müssen Sie aufrufen, um den Integerindex der Spalte 'M' zu ermitteln?
10. Welche Funktion müssen Sie aufrufen, um den Stringnamen der Spalte 14 zu ermitteln?
11. Wie können Sie ein Tupel aller Cell-Objekte von A1 bis F1 abrufen?
12. Wie speichern Sie eine Arbeitsmappe unter dem Dateinamen `example.xlsx`?
13. Wie fügen Sie in eine Zelle eine Formel ein?
14. Was müssen Sie als Erstes tun, um das Ergebnis der Formel in einer Zelle abzurufen statt im der Formel als solcher?
15. Wie setzen Sie die Höhe von Zeile 5 auf 100?
16. Wie können Sie Spalte C verstecken?
17. Was lädt OpenPyXL 2.1.4 nicht aus einer Arbeitsmappendatei?
18. Was ist ein fixierter Bereich?
19. Welche fünf Funktionen und Methoden müssen Sie aufrufen, um ein Balkendiagramm zu erstellen?

## Übungsprojekte

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erledigen:

### Multiplikationstabellen erstellen

Schreiben Sie das Programm *multiplicationTable.py*, das an der Befehlszeile eine Zahl *N* entgegennimmt und eine *N* × *N*-Multiplikationstabelle als Excel-Arbeitsblatt erstellt.

Betrachten Sie als Beispiel folgenden Aufruf des Programms:

```
py multiplicationTable.py 6
```

Daraus soll das Programm das Arbeitsblatt aus Abb. 12–11 machen.

	A	B	C	D	E	F	G	H
1	1	2	3	4	5	6		
2	1	2	3	4	5	6		
3	2	4	6	8	10	12		
4	3	6	9	12	15	18		
5	4	8	12	16	20	24		
6	5	10	15	20	25	30		
7	6	12	18	24	30	36		
8								
9								

**Abb. 12–11** Eine Multiplikationstabelle in einem Arbeitsblatt

Zeile 1 und Spalte A dienen als Beschriftungen und sollen fett dargestellt werden.

### Leere Zeilen einfügen

Das Programm *blankRowInserter.py* soll an der Befehlszeile zwei Integer und einen Dateinamenstring entgegennehmen, wobei wir den ersten Integer *N* und den zweiten *M* nennen. Beginnend mit Zeile *N* soll das Programm nun *M* leere Zeilen in das Arbeitsblatt einfügen. Betrachten Sie als Beispiel folgenden Programmaufruf:

```
python blankRowInserter.py 3 2 myProduce.xlsx
```

In Abb. 12–12 sehen Sie, wie das Arbeitsblatt vor und nach der Programmausführung aussieht.

Potatoes					
A	B	C	D	E	F
1 Potatoes	Celery	Ginger	Yellow per	Green beans	Fava beans
2 Okra	Okra	Corn	Garlic	Tomatoes	Yellow
3 Fava bean: Spinach	Grapefruit	Grapes	Apricots	Papaya	
4 Watermelon	Cucumber	Ginger	Watermelon	Red onion	Butter
5 Garlic	Apricots	Eggplant	Cherries	Strawberry	Apricot
6 Parsnips	Okra	Cucumber	Apples	Grapes	Avocado
7 Asparagus	Fava bean: Green cab	Grapefruit	Ginger	Butter	
8 Avocados	Watermelon	Eggplant	Grapes	Strawberry	Celery

Potatoes					
A	B	C	D	E	F
1 Potatoes	Celery	Ginger	Yellow per	Green beans	Fava beans
2 Okra	Okra	Corn	Garlic	Tomatoes	Yellow
3					
4					
5 Fava bean: Spinach	Grapefruit	Grapes	Apricots	Papaya	
6 Watermelon	Cucumber	Ginger	Watermelon	Red onion	Butter
7 Garlic	Apricots	Eggplant	Cherries	Strawberry	Apricot
8 Parsnips	Okra	Cucumber	Apples	Grapes	Avocado

**Abb. 12–12** Das Arbeitsblatt bevor (links) und nachdem (rechts) das Programm ab Zeile 3 zwei Leerzeilen eingefügt hat

Lassen Sie das Programm zunächst den Inhalt des Arbeitsblatts lesen. Wenn Sie das neue Arbeitsblatt schreiben, kopieren Sie die ersten N Zeilen mithilfe einer for-Schleife. Addieren Sie für die folgenden Zeilen im neuen Arbeitsblatt M zur Zeilennummer.

### Zellen transponieren

Schreiben Sie ein Programm, das Zeile und Spalte der Zellen in einem Arbeitsblatt vertauscht. Beispielsweise soll der Wert in Zeile 5, Spalte 3 in Zeile 3, Spalte 5 verlegt werden (und umgekehrt). Alle Zellen in dem Arbeitsblatt sollen auf diese Weise umgestellt werden. Abb. 12–13 zeigt ein Vorher/Nachher-Beispiel.

ITEM										
A	B	C	D	E	F	G	H	I	J	
1 ITEM	SOLD									
2 Eggplant		334								
3 Cucumber		252								
4 Green cab		238								
5 Eggplant		516								
6 Garlic		98								
7 Parsnips		16								
8 Asparagus		335								
9 Avocados		84								
10										

ITEM										
A	B	C	D	E	F	G	H	I	J	
1 ITEM	Eggplant	Cucumber	Green cab	Eggplant	Garlic	Parsnips	Asparagus	Avocados		
2 SOLD	334	252	238	516	98	16	335	84		
3										
4										
5										
6										
7										
8										
9										
10										

**Abb. 12–13** Das Arbeitsblatt vor (oben) und nach (unten) der Transponierung

Hierzu können Sie verschachtelte for-Schleifen verwenden, um die Daten im Arbeitsblatt zu lesen und in eine Liste aus Listen zu schreiben. Diese Datenstruktur enthält also beispielsweise einen Eintrag wie `sheetData[x][y]` für die Zelle in

Spalte x und Zeile y. Um das neue Arbeitsblatt zu schreiben, verwenden Sie dann sheetData[y][x] für die Zelle in Spalte x und Zeile y.

### **Textdateien in Arbeitsblätter umwandeln**

Schreiben Sie ein Programm, das den Inhalt mehrerer Textdateien liest (die Sie selbst anlegen können) und in ein Arbeitsblatt einfügt. Dabei soll eine Textzeile jeweils in eine Arbeitsblattzeile geschrieben werden, wobei die Zeilen der ersten Textdatei in Spalte A stehen, die Zeilen der zweiten Textdatei in Spalte B usw.

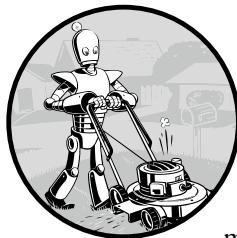
Verwenden Sie die Methode `readlines()` des `File`-Objekts, um eine Liste von Strings zurückzugeben, in der jeder String für eine Textzeile steht. Geben Sie die erste Zeile der ersten Datei in Spalte 1, Zeile 1 aus, die zweite Zeile in Spalte 1, Zeile 2 usw. Die nächste Datei, die Sie mit `readlines()` lesen, kommt dann in Spalte 2, die übernächste in Spalte 3 usw.

### **Arbeitsblätter in Textdateien umwandeln**

Schreiben Sie ein Programm, das die gegenteilige Aufgabe des vorherigen ausführt: Es soll ein Arbeitsblatt öffnen und den Inhalt der Zellen von Spalte A in eine Textdatei schreiben, den Inhalt von Spalte B in eine weitere Textdatei usw.

# 13

## Arbeiten mit PDF- und Word-Dokumenten



PDF- und Word-Dokumente sind Binärdateien, was den Umgang mit ihnen viel komplizierter macht als die Verwendung von reinen Textdateien. Neben Text enthalten sie auch eine Menge an Informationen über Schriftarten, Farben und das Layout. Wenn Ihre Programme PDF- oder Word-Dokumente lesen oder schreiben können sollen, müssen Sie mehr tun, als einfach nur den Dateinamen an `open()` zu übergeben.

Zum Glück gibt es Python-Module, die die Arbeit mit PDF- und Word-Dokumenten erleichtern. In diesem Kapitel lernen Sie zwei dieser Module kennen, nämlich PyPDF2 und Python-Docx.

### PDF-Dokumente

PDF steht für *Portable Document Format*. PDF-Dokumente tragen die Endung `.pdf` und bieten viele verschiedene Möglichkeiten, doch in diesem Kapitel beschränken wir uns auf die beiden Aufgaben, die am häufigsten ausgeführt werden: das Lesen von Text in PDFs und das Erstellen neuer PDFs aus vorhandenen Dokumenten.

Für die Arbeit mit PDFs verwenden wir das Modul PyPDF2. Um es zu installieren, führen Sie an der Befehlszeile `pip install PyPDF2` aus. Bei dem Namen wird zwischen Groß- und Kleinschreibung unterschieden; achten Sie daher genau darauf, dass Sie ein kleines `y` und sonst nur Großbuchstaben eingeben. (Einzelheiten über die Installation von Drittanbietermodulen finden Sie in Anhang A.) Wenn Sie anschließend `import PyPDF2` in der interaktiven Shell ausführen können, ohne dass eine Fehlermeldung angezeigt wird, wurde das Modul korrekt installiert.

### PDF – ein problematisches Format

PDF-Dateien bilden zwar eine großartige Möglichkeit, um Texte so zu verbreiten, dass andere sie ohne Schwierigkeiten in der vorgesehenen Gestaltung lesen und ausdrucken können, allerdings lässt sich dieses Format von Programmen nicht so einfach analysieren wie reiner Text. Daher ist es möglich, dass PyPDF2 beim Entnehmen von Text aus einem PDF Fehler macht oder manche PDFs gar nicht erst öffnen kann. Leider können Sie in einem solchen Fall nichts tun, um das Problem zu lösen – wenn PyPDF2 mit einer bestimmten PDF-Datei nicht arbeiten kann, dann ist das leider so. Allerdings ist mir bis heute noch keine PDF-Datei begegnet, die PyPDF2 nicht öffnen konnte.

### Text aus PDFs entnehmen

PyPDF2 bietet keine Möglichkeit, um aus PDF-Dokumenten Bilder, Diagramme oder andere Medien zu entnehmen, allerdings kann es Texte daraus extrahieren und als Python-String zurückgeben. Um uns anzusehen, wie PyPDF2 funktioniert, verwenden wir das Beispiel-PDF aus Abb. 13–1.

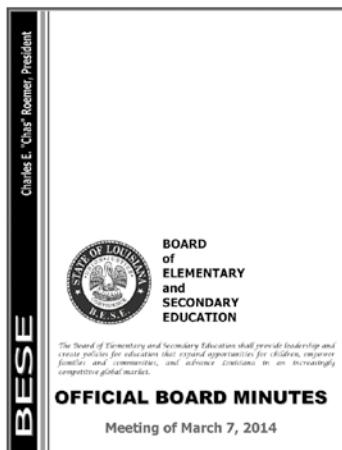


Abb. 13–1 Die PDF-Seite, aus der wir Text entnehmen

Laden Sie dieses PDF von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter und geben Sie Folgendes in die interaktive Shell ein:

```
>>> import PyPDF2  
>>> pdfFileObj = open('meetingminutes.pdf', 'rb')  
>>> pdfReader = PyPDF2.PdfFileReader(pdfFileObj)  
>>> pdfReader.numPages ❶  
19  
>>> pageObj = pdfReader.getPage(0) ❷  
>>> pageObj.extractText() ❸  
'OOFFFFIICIIAALL BBOOAARRDD MMIINNUUTTEESS Meeting of March 7, 2015  
\n The Board of Elementary and Secondary Education shall provide leadership  
and create policies for education that expand opportunities for children,  
empower families and communities, and advance Louisiana in an increasingly  
competitive global market. BOARD of ELEMENTARY and SECONDARY EDUCATION '
```

Als Erstes importieren wir hier das Modul PyPDF2. Danach öffnen wir die Datei *meetingminutes.pdf* im binären Lesemodus und speichern sie in *pdfFileObj*. Um ein PdfFileReader-Objekt für dieses PDF zu bekommen, rufen Sie die Funktion *PyPDF2.PdfFileReader()* auf und übergeben ihr *pdfFileObj*. Anschließend speichern Sie das PdfFileReader-Objekt in *pdfReader*.

Die Gesamtzahl der Seiten in dem Dokument steht im Attribut *numPages* des PdfFileReader-Objekts (❶). Unser Beispiel-PDF hat 19 Seiten, aber wir wollen hier nur Text von der ersten Seite entnehmen.

Dazu benötigen wir ein Page-Objekt, das für eine einzelne Seite steht. Um ein solches Objekt zu gewinnen, rufen Sie die Methode *getPage()* (❷) für das PdfFileReader-Objekt auf und übergeben ihr die Nummer der gewünschten Seite, in unserem Fall 0.

Der Seitenindex von PyPDF2 beginnt bei 0, d. h., die erste Seite ist Seite 0, die zweite ist Seite 1 usw. Das ist immer der Fall, auch wenn im PDF ganz andere Seitenzahlen verwendet werden. Beispielsweise kann es sein, dass ein PDF, das einen Auszug aus einem längeren Bericht darstellt, nur die Seiten 42 bis 44 enthält. Um an die erste Seite dieses Dokuments zu gelangen, müssen Sie *pdfReader.getPage(0)* aufrufen und nicht *getPage(42)* oder *getPage(1)*.

Wenn Ihnen ein Page-Objekt vorliegt, rufen Sie seine *extractText()*-Methode auf, die einen String mit dem Text der Seite zurückgibt (❸). Die Textentnahme ist nicht perfekt. In dem zurückgegebenen String in unserem Beispiel fehlt die Angabe *Charles E. "Chas" Roemer, President*, und die Spatierung ist etwas kurios. Trotzdem kann diese Annäherung an den Textinhalt des PDFs für die Zwecke Ihres Programms ausreichend sein.

## PDFs entschlüsseln

Manche PDFs sind verschlüsselt, sodass sie nur von jemandem geöffnet und gelesen werden können, der das richtige Passwort kennt. Das PDF *encrypted.pdf* aus dem Download ist beispielsweise mit dem Passwort *rosebud* verschlüsselt. Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> import PyPDF2
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
>>> pdfReader.isEncrypted ❶
True
>>> pdfReader.getPage(0)
Traceback (most recent call last): ❷
  File "<pyshell#173>", line 1, in <module>
    pdfReader.getPage()
-- snipp --
  File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173, in get0bject
    raise utils.PdfReadError("file has not been decrypted")
PyPDF2.utils.PdfReadError: file has not been decrypted
>>> pdfReader.decrypt('rosebud') ❸
1
>>> page0bj = pdfReader.getPage(0)
```

Alle PdfFileReader-Objekte verfügen über das Attribut `isEncrypted`. Wenn die PDF-Datei verschlüsselt ist, hat dieses Attribut den Wert `True`, anderenfalls ist es `False` (❶). Bevor die Datei mit dem richtigen Passwort entschlüsselt wurde, schlagen alle Versuche fehl, eine Funktion aufzurufen, die in der Datei liest (❷).

Um ein verschlüsseltes PDF zu lesen, müssen Sie die Funktion `decrypt()` aufrufen und ihr das Passwort als String übergeben (❸). Danach führt der Aufruf von `getPage()` nicht mehr zu einem Fehler. Sollten Sie jedoch das falsche Passwort bereitstellen, gibt `decrypt()` den Wert `0` zurück und es ist danach immer noch nicht möglich, `getPage()` aufzurufen. Beachten Sie, dass `decrypt()` nur das PdfFileReader-Objekt entschlüsselt und nicht die eigentliche PDF-Datei. Wenn das Programm beendet ist, bleibt die Datei auf Ihrer Festplatte verschlüsselt. Wenn Sie Ihr Programm erneut ausführen, muss es `decrypt()` ein weiteres Mal aufrufen.

## PDFs erstellen

Das Gegenstück zu den PdfFileReader-Objekten sind die PdfFileWriter-Objekte, mit denen Sie neue PDF-Dateien erstellen können. Allerdings ist PyPDF2 nicht in der Lage, willkürlichen Text in ein PDF zu verwandeln. Diese Möglichkeit beschränkt sich darauf, Seiten aus anderen PDFs zu kopieren, Seiten zu drehen, Seiten zu überlagern und Dateien zu verschlüsseln.

Mit PyPDF2 können Sie ein PDF nicht direkt bearbeiten. Stattdessen müssen Sie ein neues PDF erstellen und dann den Inhalt aus einem vorhandenen Dokument dort hineinkopieren. Für die Beispiele in diesem Abschnitt gilt die folgende allgemeine Vorgehensweise:

1. Öffnen Sie ein oder mehrere vorhandene PDFs (die Quell-PDFs) in PdfFileReader-Objekten.
2. Erstellen Sie ein neues PdfFileWriter-Objekt.
3. Kopieren Sie Seiten aus den PdfFileReader-Objekten in das PdfFileWriter-Objekt.
4. Verwenden Sie abschließend das PdfFileWriter-Objekt, um das Ausgabe-PDF zu schreiben.

Wenn Sie ein PdfFileWriter-Objekt erstellen, wird dadurch nur ein Wert angelegt, der in Python für ein PDF-Dokument steht. Die eigentliche PDF-Datei wird dadurch nicht erstellt. Dazu müssen Sie die PdfFileWriter-Methode `write()` aufrufen.

Die Methode `write()` nimmt ein reguläres File-Objekt entgegen, das im binären Schreibmodus geöffnet wurde. Ein solches Objekt erhalten Sie, indem Sie die Python-Funktion `open()` mit zwei Argumenten aufrufen, nämlich dem String mit dem gewünschten Dateinamen des PDFs und der Angabe 'wb' für den binären Schreibmodus.

Machen Sie sich keine Sorgen, wenn das ein bisschen verwirrend klingen sollte – in den folgenden Codebeispielen werden Sie genau sehen, wie es funktioniert.

## Seiten kopieren

Mit PyPDF2 können Sie Seiten von einem PDF-Dokument in ein anderes kopieren. Damit ist es möglich, mehrere PDF-Dateien zu kombinieren und dabei ungewünschte Seiten wegzulassen oder die Reihenfolge der Seiten zu ändern.

Laden Sie `meetingminutes.pdf` und `meetingminutes2.pdf` von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter und stellen Sie diese Dateien in das aktuelle Arbeitsverzeichnis. Geben Sie dann Folgendes in die interaktive Shell ein:

```
>>> import PyPDF2  
>>> pdf1File = open('meetingminutes.pdf', 'rb')  
>>> pdf2File = open('meetingminutes2.pdf', 'rb')  
>>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File) ❶  
>>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File) ❷  
>>> pdfWriter = PyPDF2.PdfFileWriter() ❸  
  
>>> for pageNum in range(pdf1Reader.numPages):  
    pageObj = pdf1Reader.getPage(pageNum) ❹  
    pdfWriter.addPage(pageObj) ❺
```

```
>>> for pageNum in range(pdf2Reader.numPages):
    pageObj = pdf2Reader.getPage(pageNum)      ④
    pdfWriter.addPage(pageObj)      ⑤

>>> pdfOutputFile = open('combinedminutes.pdf', 'wb')      ⑥
>>> pdfWriter.write(pdfOutputFile)
>>> pdfOutputFile.close()
>>> pdf1File.close()
>>> pdf2File.close()
```

Öffnen Sie beide PDF-Dateien im binären Lesemodus und speichern Sie die beiden resultierenden File-Objekte in pdf1File und pdf2File. Rufen Sie dann die Funktion PyPDF2.PdfFileReader() auf und übergeben Sie ihr pdf1File, um das PdfFileReader-Objekt für *meetingminutes.pdf* zu bekommen (①). In einem zweiten Aufruf übergeben Sie pdf2File für *meetingminutes2.pdf* (②). Erstellen Sie dann ein neues PdfFileWriter-Objekt, das zunächst für ein leeres PDF-Dokument steht (③).

Als Nächstes kopieren Sie alle Seiten der beiden Quell-PDFs und fügen Sie zu dem PdfFileWriter-Objekt hinzu. Rufen Sie getPage() für eines der PdfFileReader-Objekte auf, um ein Page-Objekt zu erhalten (④), und übergeben Sie dieses Objekt an die PdfFileWriter-Methode addPage() (⑤). Diese Schritte führen Sie erst für pdf1Reader und dann noch einmal für pdf2Reader aus. Wenn die Kopierarbeiten erledigt sind, schreiben Sie das neue PDF *combinedminutes.pdf*, indem Sie der PdfFileWriter-Methode write() ein File-Objekt übergeben (⑥).

### Hinweis

PyPDF2 ist nicht in der Lage, Seiten in der Mitte eines PdfFileWriter-Objekts einzufügen. Die Methode addPage() hängt Seiten immer nur am Ende an.

Damit haben Sie eine neue PDF-Datei erstellt, die die Seiten aus *meetingminutes.pdf* und *meetingminutes2.pdf* in einem einzigen Dokument kombiniert. Denken Sie daran, dass Sie das an PyPDF2.PdfFileReader() übergebene File-Objekt im binären Lesemodus öffnen müssen, indem Sie 'rb' als zweites Argument an open() übergeben. Ebenso müssen Sie das an PyPDF2.PdfFileWriter() übergebene File-Objekt mithilfe von 'wb' im binären Schreibmodus öffnen.

### Seiten drehen

Mit den Methoden rotateClockwise() und rotateCounterClockwise() können Sie die Seiten eines PDFs in Schritten von 90° im bzw. gegen den Uhrzeigersinn drehen. Übergeben Sie diesen Methoden dazu die Ganzzahlen 90, 180 oder 270. Probieren Sie das wie folgt in der interaktiven Shell aus (wobei sich *meetingsminutes.pdf* im aktuellen Arbeitsverzeichnis befinden muss):

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
>>> page = pdfReader.getPage(0)    ❶
>>> page.rotateClockwise(90)      ❷
{'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
-- schnipp --
}
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(page)
>>> resultPdfFile = open('rotatedPage.pdf', 'wb')    ❸
>>> pdfWriter.write(resultPdfFile)
>>> resultPdfFile.close()
>>> minutesFile.close()
```

Hier wählen wir die erste Seite des PDFs mit `getPage(0)` aus (❶) und wenden dann `rotateClockwise(90)` darauf an (❷). Anschließend schreiben wir ein neues PDF mit der gedrehten Seite und speichern es als `rotatedPage.pdf` (❸).

In dem resultierenden PDF ist die erste Seite um 90° im Uhrzeigersinn gedreht (siehe Abb. 13–2). Die Rückgabewerte von `rotateClockwise()` und `rotateCounterClockwise()` enthalten eine Menge Informationen, die Sie aber getrost ignorieren können.

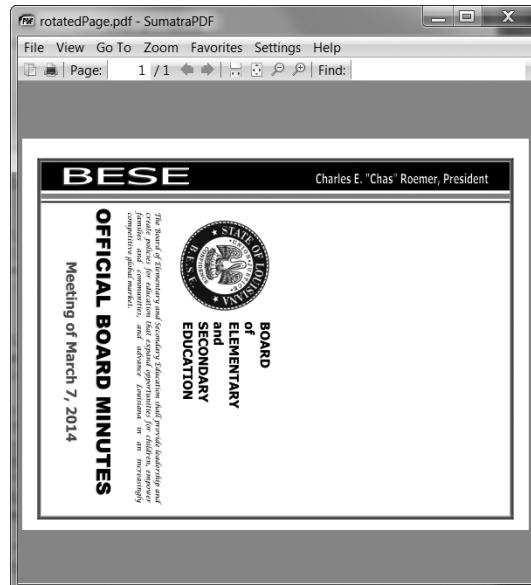


Abb. 13–2 Die Datei `rotatedPage.pdf` mit der um 90° im Uhrzeigersinn gedrehten Seite

## Seiten überlagern

PyPDF2 kann den Inhalt einer Seite auch mit dem einer anderen überlagern, was eine praktische Einrichtung ist, um Seiten mit einem Logo, einem Zeitstempel oder einem Wasserzeichen zu versehen. Mit Python können Sie sehr leicht in mehreren Dateien Wasserzeichen gezielt zu bestimmten Seiten hinzufügen.

Laden Sie *watermark.pdf* von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter und platzieren Sie die Datei neben *meetingminutes.pdf* im aktuellen Arbeitsverzeichnis. Geben Sie dann folgenden Code in die interaktive Shell ein:

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)    ❶
>>> minutesFirstPage = pdfReader.getPage(0)    ❷
>>> pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
❸
>>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))    ❹
>>> pdfWriter = PyPDF2.PdfFileWriter()    ❺
>>> pdfWriter.addPage(minutesFirstPage)    ❻
❻
>>> for pageNum in range(1, pdfReader.numPages):    ❼
    pageObj = pdfReader.getPage(pageNum)
    pdfWriter.addPage(pageObj)

>>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> minutesFile.close()
>>> resultPdfFile.close()
```

Hier erstellen wir zunächst ein PdfFileReader-Objekt von *meetingminutes.pdf* (❶). Um ein Page-Objekt für die erste Seite zu erhalten, rufen wir getPage(0) auf und speichern das resultierende Objekt in *minutesFirstPage* (❷). Anschließend erstellen wir ein PdfFileReader-Objekt für *watermark.pdf* (❸) und rufen mergePage() für *minutesFirstPage* auf (❹). Als Argument für mergePage() übergeben wir dabei das Page-Objekt für die erste Seite von *watermark.pdf*.

Danach stellt *minutesFirstPage* die erste Seite einschließlich des Wasserzeichens dar. Nun legen wir ein PdfFileWriter-Objekt an (❺) und fügen ihm die erste Seite mit Wasserzeichen hinzu (❻). Anschließend durchlaufen wir den Rest der Seiten in *meetingminutes.pdf* und fügen auch sie zu dem PdfFileWriter-Objekt hinzu (❼). Als Letztes öffnen wir ein neues PDF namens *watermarkedCover.pdf* und schreiben den Inhalt des PdfFileWriter-Objekts hinein.

Das Ergebnis sehen Sie in Abb. 13–3. Das neue PDF *watermarkedCover.pdf* enthält den gesamten Inhalt von *meetingminutes.pdf*, wobei die erste Seite jedoch mit einem Wasserzeichen versehen ist.



**Abb. 13-3** Das ursprüngliche PDF (links), das PDF für das Wasserzeichen (Mitte) und das kombinierte PDF (rechts)

## PDFs verschlüsseln

Ein PdfFileWriter-Objekt kann ein PDF-Dokument auch mit einer Verschlüsselung versehen:

```
>>> import PyPDF2
>>> pdfFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdfReader.numPages):
    pdfWriter.addPage(pdfReader.getPage(pageNum))

>>> pdfWriter.encrypt('swordfish') ❶
>>> resultPdf = open('encryptedminutes.pdf', 'wb')
>>> pdfWriter.write(resultPdf)
>>> resultPdf.close()
```

Bevor wir die Datei mit `write()` speichern, rufen wir die Methode `encrypt()` auf und übergeben ihr den String für das gewünschte Passwort (❶). PDFs können über ein *Benutzerpasswort* und ein *Besitzerpasswort* verfügen. Wenn Sie das Benutzerpasswort eingeben, dürfen Sie das PDF lesen; mit dem Besitzerpasswort sind Sie darüber hinaus in der Lage, Berechtigungen für das Drucken, das Kommentieren, die Textentnahme und andere Vorgänge festzulegen. Diese Passwörter werden als das erste bzw. zweite Argument an `encrypt()` übergeben. Ist nur ein Argument vorhanden, wird es für beide Passwörter benutzt.

In dem vorstehenden Beispiel haben wir die Seiten von *meetingminutes.pdf* in ein PdfFileWriter-Objekt kopiert. Dieses Objekt haben wir anschließend mit dem Passwort *swordfish* geschützt und dann in das neue PDF *encryptedminutes.pdf*

kopiert. Bevor jemand *encryptedminutes.pdf* einsehen kann, muss er erst das Passwort eingeben. Nachdem Sie sich vergewissert haben, dass die Kopie korrekt ist, können Sie die ursprüngliche, unverschlüsselte Datei *meetingminutes.pdf* löschen.

## Projekt: Ausgewählte Seiten aus mehreren PDFs kombinieren

Stellen Sie sich vor, Sie haben die langweilige Aufgabe, mehrere Dutzend PDF-Dokumente zu einer einzigen Datei zu kombinieren. Die erste Seite dieser Einzeldateien ist jeweils ein Deckblatt, das allerdings in der Gesamtdatei nicht wiederholt vorkommen soll. Es gibt zwar eine ganze Menge kostenloser Programme, um PDFs zu kombinieren, aber viele davon verketten einfach nur komplett Dateien. Schreiben wir daher ein Python-Programm, bei dem Sie angeben können, welche Seiten in dem kombinierten PDF vorhanden sein sollen.

Grob gesagt, soll das Programm Folgendes tun:

- Alle PDF-Dateien im aktuellen Arbeitsverzeichnis finden
- Die PDFs nach Dateinamen sortieren, sodass sie in der richtigen Reihenfolge kombiniert werden
- Jede Seite aller PDFs außer der jeweils ersten in die Ausgabedatei schreiben

Der Code muss daher Folgendes tun:

- Mit `os.listdir()` alle Dateien im Arbeitsverzeichnis finden und alle Nicht-PDF-Dateien ausschließen
- Die Dateien mithilfe der Python-Listenmethode `sort()` alphabetisch nach ihren Namen sortieren
- Ein `PdfFileWriter`-Objekt für das Ausgabe-PDF erstellen
- Alle PDF-Dateien durchlaufen und jeweils ein `PdfFileReader`-Objekt davon erstellen
- Alle Seiten (außer jeweils der ersten) aller Dateien durchlaufen
- Die Seiten zum Ausgabe-PDF hinzufügen
- Das Ausgabe-PDF in die Datei *allminutes.pdf* schreiben

Für dieses Projekt öffnen Sie ein neues Dateieditorfenster und speichern das noch leere Programm als *combinePdfs.py*.

### Schritt 1: Alle PDF-Dateien finden

Als Erstes muss Ihr Programm eine Liste aller Dateien mit der Endung *.pdf* im aktuellen Arbeitsverzeichnis aufstellen und sortieren. Geben Sie dazu folgenden Code ein:

```
#! python3
# combinePdfs.py - Kombiniert alle PDFs im aktuellen Arbeitsverzeichnis zu ' '
# einem einzigen PDF

import PyPDF2, os ❶

# Ruft die Namen aller PDF-Dateien ab
pdfFiles = []
for filename in os.listdir('.'):
    if filename.endswith('.pdf'):
        pdfFiles.append(filename) ❷
pdfFiles.sort(key=str.lower) ❸

pdfWriter = PyPDF2.PdfFileWriter() ❹

# TODO: Alle PDF-Dateien durchlaufen

# TODO: Alle Seiten (außer der ersten) durchlaufen und hinzufügen

# TODO: Das resultierende PDF in einer Datei speichern
```

Nach der Shebang-Zeile und dem Kommentar mit der Beschreibung des Programms importiert der Code die Module `os` und `PyPDF2` (❶). Der Aufruf von `os.listdir('.')` gibt eine Liste aller Dateien im aktuellen Arbeitsverzeichnis zurück. Danach durchläuft der Code diese Liste und fügt nur die Dateien mit der Endung `.pdf` zu `pdfFiles` hinzu (❷). Diese Liste wird daraufhin mit dem Schlüsselwortargument `key=str.lower` von `sort()` alphabetisch geordnet (❸).

Schließlich wird ein `PdfFileWriter`-Objekt erstellt, um die kombinierten PDF-Seiten aufzunehmen (❹). Die Kommentare am Ende skizzieren die noch fehlenden Teile des Programms.

## Schritt 2: Die einzelnen PDFs öffnen

Als Nächstes muss das Programm die PDF-Dateien in `pdfFiles` lesen. Ergänzen Sie den Code wie folgt:

```
#! python3
# combinePdfs.py - Kombiniert alle PDFs im aktuellen Arbeitsverzeichnis zu ' '
# einem einzigen PDF

import PyPDF2, os

# Ruft die Namen aller PDF-Dateien ab
pdfFiles = []

-- schnipp --
```

```

# Durchläuft alle PDF-Dateien
for filename in pdfFiles:
    pdfFileObj = open(filename, 'rb')
    pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
    # TODO: Alle Seiten (außer der ersten) durchlaufen und hinzufügen

    # TODO: Das resultierende PDF in einer Datei speichern

```

Die Schleife öffnet alle PDF-Dateien im binären Lesemodus, indem sie `open()` mit `'rb'` als zweitem Argument aufruft. Diese Funktion gibt jeweils ein File-Objekt zurück, das an `PyPDF2.PdfFileReader()` übergeben wird, um ein PdfFileReader-Objekt für die betreffende PDF-Datei anzulegen.

### Schritt 3: Die einzelnen Seiten hinzufügen

In jedem PDF müssen alle Seiten außer der ersten durchlaufen werden. Ergänzen Sie das Programm mit folgendem Code:

```

#!/usr/bin/python3
# combinePdfs.py - Kombiniert alle PDFs im aktuellen Arbeitsverzeichnis zu
# einem einzigen PDF

import PyPDF2, os

-- schnipp --

# Durchläuft alle PDF-Dateien
for filename in pdfFiles:
-- schnipp --
    # Durchläuft alle Seiten (außer der ersten) und fügt sie hinzu
    for pageNum in range(1, pdfReader.numPages): ❶
        pageObj = pdfReader.getPage(pageNum)
        pdfWriter.addPage(pageObj)

    # TODO: Das resultierende PDF in einer Datei speichern

```

Der Code innerhalb der `for`-Schleife kopiert jedes Page-Objekt einzeln in das `PdfFileWriter`-Objekt. Da wir jeweils die erste Seite überspringen wollen und `PyPDF2` der ersten Seite die Nummer 0 gibt, muss der Wert des Schleifenzählers bei 1 beginnen (❶) und bis zu dem Integer in `pdfReader.numPages` gehen, diesen aber nicht einschließen.

## Schritt 4: Die Ergebnisse speichern

Nachdem die verschachtelten Schleifen durchlaufen sind, enthält die Variable `pdfWriter` ein `PdfFileWriter`-Objekt mit den kombinierten Seiten aller PDFs. Der letzte Schritt besteht darin, diesen Inhalt in eine Datei auf der Festplatte zu schreiben. Fügen Sie dem Programm folgenden Code hinzu:

```
#! python3
# combinePdfs.py - Kombiniert alle PDFs im aktuellen Arbeitsverzeichnis zu 'onefile.pdf'
# einem einzigen PDF

import PyPDF2, os

-- schnipp --

# Durchläuft alle PDF-Dateien
for filename in pdfFiles:
    -- schnipp --
        # Durchläuft alle Seiten (außer der ersten) und fügt sie hinzu
        for pageNum in range(1, pdfReader.numPages):
            -- schnipp --

# Speichert das resultierende PDF in einer Datei
pdfOutput = open('allminutes.pdf', 'wb')
pdfWriter.write(pdfOutput)
pdfOutput.close()
```

Dadurch, dass Sie 'wb' an `open()` übergeben, öffnen Sie die PDF-Ausgabedatei *allminutes.pdf* im binären Schreibmodus. Anschließend übergeben Sie das resultierende `File`-Objekt an die Methode `write()`, um die PDF-Datei zu schreiben. Ein Aufruf von `closes()` beendet das Programm.

## Vorschläge für ähnliche Programme

Mit der Möglichkeit, PDFs aus den Seiten anderer PDFs zusammenzustellen, können Sie Programme für folgende Aufgaben schreiben:

- Einzelne Seiten aus PDFs herausnehmen
- Seiten in einem PDF umordnen
- Ein PDF aus Seiten zusammenstellen, die einen bestimmten, durch `extractText()` bestimmten Text enthalten

## Word-Dokumente

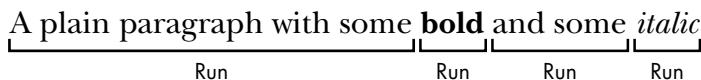
Mit dem Modul Python-Docx kann Python auch Word-Dokumente mit der Endung `.docx` erstellen und bearbeiten. Um das Modul zu installieren, führen Sie `pip install python-docx` aus. (Einzelheiten zur Installation von Drittanbietermodulen erfahren Sie in Anhang A.)

### Hinweis

Achten Sie bei der Installation darauf, dass Sie `python-docx` angeben und nicht `docx`. Dies ist der Installationsname für ein anderes Modul, das in diesem Buch nicht behandelt wird. Zum Importieren dagegen müssen Sie `import docx` schreiben, nicht `import python-docx`.

Neben Word können auch die kostenlosen alternativen Anwendungen LibreOffice Writer und OpenOffice Writer für Windows, OS X und Linux `.docx`-Dateien öffnen. Diese Software können Sie von <https://www.libreoffice.org> bzw. <http://openoffice.org> herunterladen. Die vollständige Dokumentation von `python-docx` finden Sie auf <https://python-docx.readthedocs.org/>. Es gibt zwar auch eine Version von Word für OS X, doch in diesem Kapitel konzentrieren wir uns auf Word für Windows.

`.docx`-Dateien weisen eine vielschichtigere Struktur auf als reine Textdateien. Dargestellt wird diese Struktur durch drei verschiedene Datentypen in Python-Docx. Auf der obersten Ebene befindet sich ein `Document`-Objekt für das gesamte Dokument, das eine Liste von `Paragraph`-Objekten für die einzelne Absätze des Dokuments enthält. (In einem Word-Dokument wird ein neuer Absatz erstellt, wenn der Benutzer bei der Texteingabe die Eingabetaste drückt.) Jedes `Paragraph`-Objekt wiederum besteht aus einer Liste von ein oder mehreren `Run`-Objekten. Der Absatz in Abb. 13–4 umfasst nur einen Satz, zerfällt aber in vier »Runs«.



**Abb. 13–4** Die Run-Objekte in einem Paragraph-Objekt

Der Text in einem Word-Objekt besteht nicht nur aus einem String, sondern enthält auch Informationen über Schriftart, Größe, Farbe und andere Gestaltungsmerkmale oder *Formatierungen*. Ein Run-Objekt ist eine durchgängige Folge von Text mit derselben Formatierung. Bei einem Formatierungswechsel beginnt ein neues Run-Objekt.

## Word-Dokumente lesen

Wir wollen nun ein wenig mit dem Modul `python-docx` experimentieren. Laden Sie `demo.docx` von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter und speichern Sie das Dokument im Arbeitsverzeichnis. Geben Sie dann Folgendes in die interaktive Shell ein:

```
>>> import docx
>>> doc = docx.Document('demo.docx')    ❶
>>> len(doc.paragraphs)    ❷
7
>>> doc.paragraphs[0].text    ❸
'Document Title'
>>> doc.paragraphs[1].text    ❹
'A plain paragraph with some bold and some italic'
>>> len(doc.paragraphs[1].runs)    ❺
4
>>> doc.paragraphs[1].runs[0].text    ❻
'A plain paragraph with some '
>>> doc.paragraphs[1].runs[1].text    ❼
'bold'
>>> doc.paragraphs[1].runs[2].text    ❽
' and some '
>>> doc.paragraphs[1].runs[3].text    ❾
'italic'
```

Bei (❶) öffnen wir eine `.docx`-Datei in Python, rufen `docx.Document()` auf und übergeben den Dateinamen `demo.docx`. Dadurch wird ein `Document`-Objekt zurückgegeben, dessen Attribut `paragraphs` eine Liste von `Paragraph`-Objekten enthält. Wenn wir `len()` für `doc.paragraphs` aufrufen, wird 7 zurückgegeben, was bedeutet, dass das Dokument sieben `Paragraph`-Objekte (also sieben Absätze) enthält (❷). Jedes dieser `Paragraph`-Objekte verfügt über das Attribut `text`, das einen String des Textes in dem Absatz (ohne Formatierungsinformationen) enthält. Das erste `text`-Attribut enthält hier den String `'DocumentTitle'` (❸), das zweite `'A plain paragraph with some bold and some italic'` (❹).

Außerdem verfügt jedes `Paragraph`-Objekt über das Attribut `runs`, das eine Liste der `Run`-Objekte angibt. Auch `Run`-Objekte haben ein `text`-Attribut, das den Text in dem betreffenden `Run` angibt. Sehen wir uns die `text`-Attribute im zweiten `Paragraph`-Objekt, `'A plain paragraph with some bold and some italic'`, genauer an. Wenn Sie für dieses Objekt `len()` aufrufen, erfahren Sie, dass es aus vier `Run`-Objekten besteht (❺). Das erste dieser `Run`-Objekte enthält den Text `'A plain paragraph with some '` (❻). Danach wechselt der Text in Fettschrift, sodass mit `'bold'` ein neues `Run`-Objekt beginnt (❼). Nach diesem Wort wird die Fettstellung

wieder aufgehoben, weshalb es ein drittes Run-Objekt mit dem Text ' and some ' gibt (❸). Das vierte und letzte Run-Objekt schließlich enthält den Text 'italic' in kursiver Formatierung (❹).

Mit Python-Docx können Ihre Python-Programme Text in .docx-Dateien lesen und wie andere Stringwerte behandeln.

### Den kompletten Text einer .docx-Datei abrufen

Wenn Sie nur an dem Text eines Word-Dokuments interessiert sind und nicht an den Formatierungsinformationen, können Sie die Funktion `getText()` verwenden. Sie nimmt den Namen einer .docx-Datei entgegen und gibt den darin enthaltenen Text als einzelnen Stringwert zurück. Geben Sie folgenden Code im Dateieditor ein und speichern Sie ihn als *readDocx.py*:

```
#! python3

import docx

def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

Die Funktion `getText()` öffnet das Word-Dokument, durchläuft alle Paragraph-Objekte in der Liste `paragraphs` und hängt deren Text an die Liste in `fullText` an. Nach der Schleife werden die Strings in `fullText` mithilfe von Zeilenumbruchszeichen verknüpft.

Das Programm *readDocx.py* können Sie wie jedes andere Modul importieren. Wenn Sie nur den Text eines Word-Dokuments brauchen, können Sie daher einfach Folgendes eingeben:

```
>>> import readDocx
>>> print(readDocx.getText('demo.docx'))
Document Title
A plain paragraph with some bold and some italic
Heading, level 1
Intense quote
first item in unordered list
first item in ordered list
```

Sie können `getText()` anpassen, um den String vor der Rückgabe zu verändern. Um beispielsweise jeden Absatz einzurücken, ersetzen Sie den Aufruf von `append()` in `readDocx.py` durch folgenden Code:

```
fullText.append(' ' + para.text)
```

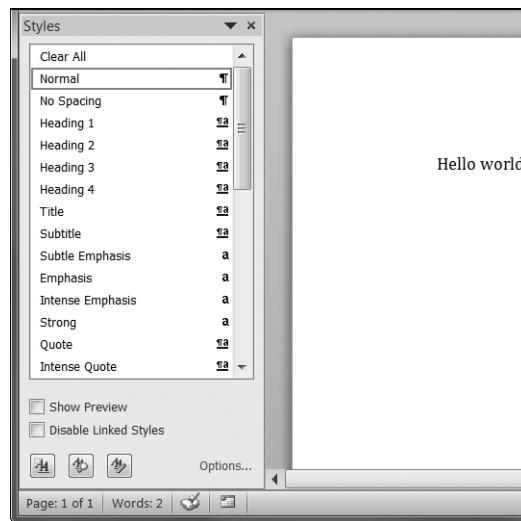
Um eine Leerzeile zwischen zwei Absätzen einzuschalten, ändern Sie die Zeile mit dem Aufruf von `join()` wie folgt ab:

```
return '\n\n'.join(fullText)
```

Wie Sie sehen, brauchen Sie nur wenige Zeilen Code, um Funktionen zu schreiben, die eine `.docx`-Datei lesen und den Inhalt als einen String nach Ihren Vorstellungen zurückgeben.

## Absätze und Run-Objekte formatieren

In Word für Windows können Sie sich die *Formate* – gespeicherte Kombinationen von Formatierungen – ansehen, indem Sie mit `Strg` + `Alt` + `Umschalt` + `S` das Dialogfeld *Formatvorlagen* öffnen (siehe Abb. 13–5). Unter OS X wählen Sie dazu im Menü *Ansicht > Formatvorlagen*.



**Abb. 13–5** Das Dialogfeld *Formatvorlagen* in Windows

In Word und anderen Textverarbeitungssystemen können Sie *Formate* verwenden, um ähnliche Arten von Text einheitlich zu gestalten und ihr Erscheinungsbild leicht ändern zu können. Wenn Sie Absätze im Fließtext in Times New Roman mit einer

Größe von 11 Punkt und in linksbündiger Ausrichtung gestalten möchten, können Sie ein Format mit diesen Einstellungen anlegen und allen entsprechenden Absätzen zuweisen. Wollen Sie später das Aussehen aller Fließtextabsätze im gesamten Dokument ändern, müssen Sie lediglich das Format anpassen und schon werden alle entsprechenden Absätze automatisch aktualisiert.

Für Word-Dokumente gibt es drei Arten von Formaten: *Absatzformate* für Paragraph-Objekte, *Zeichenformate* für Run-Objekte und *verknüpfte Formate* für beide Arten von Objekten. Um Paragraph- und Run-Objekten Formate zuweisen, geben Sie in ihrem style-Attribut den String mit dem Namen des gewünschten Formats an. Wenn style den Wert None hat, ist mit dem Objekt kein Format verbunden.

Die Standardformate von Word haben folgende Stringwerte:

```
'Normal'  
'BodyText'  
'BodyText2'  
'BodyText3'  
'Caption'  
'Heading1'  
'Heading2'  
'Heading3'  
'Heading4'  
'Heading5'  
'Heading6'  
'Heading7'  
'Heading8'  
'Heading9'  
'IntenseQuote'  
'List'  
'List2'  
'List3'  
'ListBullet'  
'ListBullet2'  
'ListBullet3'  
'ListContinue'  
'ListContinue2'  
'ListContinue3'  
'ListNumber'  
'ListNumber2'  
'ListNumber3'  
'ListParagraph'  
'MacroText'  
'NoSpacing'  
'Quote'  
'Subtitle'  
'TOCHeading'  
'Title'
```

Die Formatnamen im style-Attribut dürfen keine Leerzeichen aufweisen. Wenn ein Format in Word auch als *Subtle Emphasis* angezeigt wird, müssen Sie im style-Attribut trotzdem den Stringwert 'SubtleEmphasis' statt 'Subtle Emphasis' verwenden. Leerzeichen führen dazu, dass Word den Formatnamen falsch liest und das Format nicht anwendet.

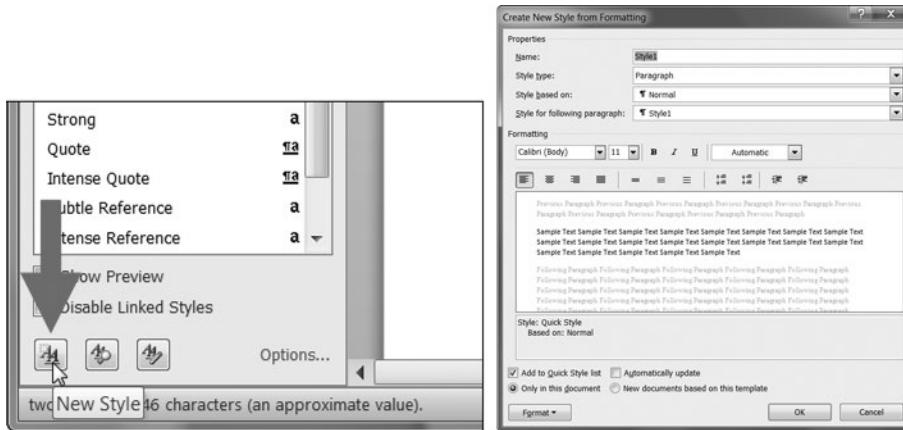
Wenn Sie für ein Run-Objekt ein verknüpftes Format verwenden, müssen Sie am Ende des Namens 'Char' anhängen. Um beispielsweise das verknüpfte Format *Quote* zu verwenden, geben Sie für ein Paragraph-Objekt paragraphObj.style = 'Quote' an, für ein Run-Objekt aber runObj.style = 'QuoteChar'.

In der aktuellen Version 0.7.4 von Python-Docx können Sie nur die Standardformate von Word und die in dem geöffneten .docx-Dokument vorhandenen Formate verwenden. Es ist nicht möglich, neue Formate anzulegen. In kommenden Versionen von Python-Docx mag sich das aber ändern.

### **Word-Dokumente mit anderen als den Standardformaten erstellen**

Wenn Sie ein Word-Dokument mit anderen als den Standardformaten erstellen wollen, müssen Sie in Word ein neues Word-Dokument öffnen und darin die Formate anlegen, indem Sie unten im Dialogfeld *Formatvorlagen* auf *Neue Formatvorlage* klicken (siehe Abb. 13–6).

Dadurch wird das Dialogfeld *Neue Formatvorlage* geöffnet, in der Sie ein neues Format gestalten können. Wenn Sie fertig sind, öffnen Sie in der interaktiven Shell das leere Dokument mit docx.Document() und verwenden Sie es als Grundlage für Ihr neues Word-Dokument. Das neue Format steht nun unter dem von Ihnen gewählten Namen in Python-Docx zur Verfügung.



**Abb. 13–6** Die Schaltfläche Neue Formatvorlage (links) und das Dialogfeld Neue Formatvorlage (rechts)

## Run-Attribute

Runs können mithilfe von Textattributen zusätzlich formatiert werden. Jedes dieser Attribute lässt sich auf drei verschiedene Werte setzen: True (das Attribut ist immer aktiviert, unabhängig davon, welche anderen Formate dem Run zugewiesen sind), False (das Attribut ist immer ausgeschaltet) und None (das betreffende Gestaltungsmerkmal erhält den Wert aus dem Format, das dem Run zugewiesen ist).

Eine Aufstellung der Textattribute für Run-Objekte finden Sie in Tabelle 13–1.

Attribut	Bedeutung
bold	Der Text wird fett dargestellt.
italic	Der Text wird kursiv dargestellt.
underline	Der Text wird unterstrichen.
strike	Der Text wird durchgestrichen.
double_strike	Der Text wird doppelt durchgestrichen.
all_caps	Der Text wird komplett in Großbuchstaben geschrieben.
small_caps	Der Text wird in Kapitälchen geschrieben, wobei die Kleinbuchstaben so aussehen wie Großbuchstaben, aber 2 Punkt kleiner sind.
shadow	Der Text wird mit einem Schatten dargestellt.
outline	Der Text wird nicht massiv dargestellt, sondern als Umrisslinie.
rtl	Der Text wird von rechts nach links geschrieben.
imprint	Der Text wirkt, als sei er in die Seite hineingeprägt.
emboss	Der Text wirkt, als trete er aus der Seite hervor.

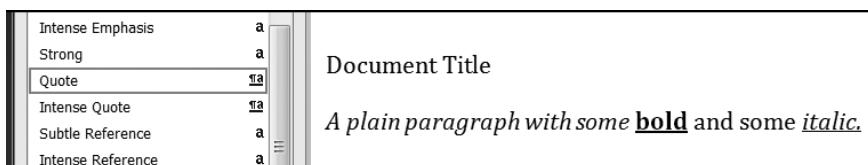
**Tab. 13–1** Textattribute für Run-Objekte

Um beispielsweise die Formatierungen in *demo.docx* zu ändern, geben Sie in der interaktiven Shell Folgendes ein:

```
>>> doc = docx.Document('demo.docx')
>>> doc.paragraphs[0].text
'Document Title'
>>> doc.paragraphs[0].style
'Title'
>>> doc.paragraphs[0].style = 'Normal'
>>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
>>> (doc.paragraphs[1].runs[0].text, doc.paragraphs[1].runs[1].text, doc.
paragraphs[1].runs[2].text, doc.paragraphs[1].runs[3].text)
('A plain paragraph with some ', 'bold', ' and some ', 'italic')
>>> doc.paragraphs[1].runs[0].style = 'QuoteChar'
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

Als Erstes verwenden wir hier die Attribute `text` und `style`, um zu sehen, welcher Text in den Absätzen unseres Dokuments enthalten ist und welche Formate sie aufweisen. Es ist auch sehr einfach, einen Absatz in Runs aufzuteilen und einzeln auf die Runs zuzugreifen. Auf diese Weise rufen wir den ersten, zweiten und vierten Run des zweiten Absatzes auf, gestalten sie und speichern das Ergebnis in einem neuen Dokument.

Der Text *Document Title* am Anfang von *restyled.docx* hat nun das Format *Normal* statt *Title*, das Run-Objekt für den Text *A plain paragraph with some* erhält das Format *QuoteChar* und das Attribut `underline` der beiden Run-Objekte für die Wörter *bold* und *italic* wird auf `True` gesetzt. Abb. 13–7 zeigt, wie die Formatierung der Absätze und Runs in *restyled.docx* aussieht.



**Abb. 13–7** Die Datei *restyled.docx*

Eine ausführlichere Dokumentation über die Verwendung von Formatierungen in Python-Docx finden Sie auf <https://python-docx.readthedocs.org/en/latest/user/styles.html>.

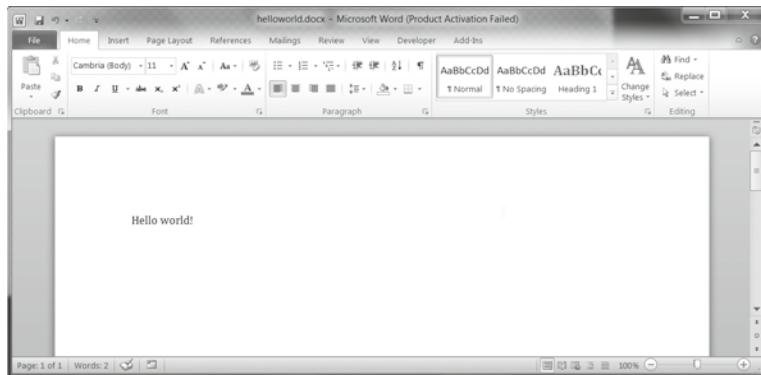
## Word-Dokumente schreiben

Geben Sie Folgendes in die interaktive Shell ein:

```
>>> import docx  
>>> doc = docx.Document()  
>>> doc.add_paragraph('Hello world!')  
<docx.text.Paragraph object at 0x0000000003B56F60>  
>>> doc.save('helloworld.docx')
```

Um selbst eine *.docx*-Datei zu erstellen, rufen Sie die Funktion `docx.Document()` auf, die ein neues, leeres Document-Objekt zurückgibt. Mit der Dokumentmethode `add_paragraph()` fügen Sie dem Dokument dann einen Absatz mit Text hinzu. Der Rückgabewert dieser Methode ist ein Verweis auf das entsprechende Paragraph-Objekt. Wenn Sie mit dem Schreiben von Text fertig sind, übergeben Sie einen String mit dem gewünschten Dateinamen an die Dokumentmethode `save()`, um das Document-Objekt als Datei zu speichern.

In dem vorstehenden Beispiel wird dadurch die Datei *helloworld.docx* im aktuellen Arbeitsverzeichnis erstellt. Wenn Sie sie öffnen, sieht sie so aus wie in Abb. 13–8.



**Abb. 13–8** Das mit `add_paragraph('Hello world!')` erstellte Word-Dokument

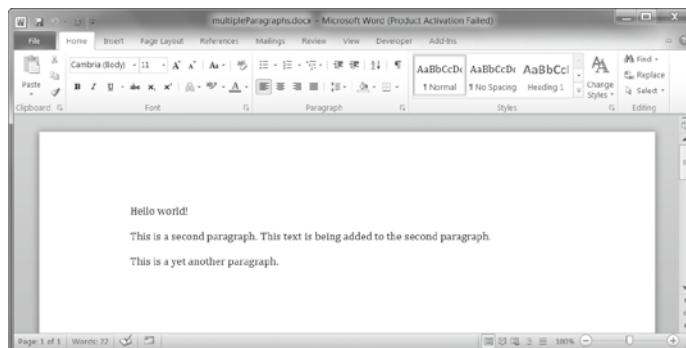
Um weitere Absätze hinzuzufügen, rufen Sie erneut die Methode `add_paragraph()` mit dem Text für den neuen Absatz auf. Sie können aber auch Text am Ende eines vorhandenen Absatzes anfügen. Dazu rufen Sie die Methode `add_run()` des Absatzes auf und übergeben Ihr einen String mit dem gewünschten Text. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x000000000366AD30>
>>> paraObj1 = doc.add_paragraph('This is a second paragraph.')
>>> paraObj2 = doc.add_paragraph('This is a yet another paragraph.')
>>> paraObj1.add_run(' This text is being added to the second paragraph.')
<docx.text.Run object at 0x0000000003A2C860>
>>> doc.save('multipleParagraphs.docx')
```

Das resultierende Dokument sehen Sie in Abb. 13–9. Der Text *This text is being added to the second paragraph.* wurde zu dem Paragraph-Objekt in paraObj1 hinzugefügt, dem zweiten Absatz, den Sie in doc geschrieben haben. Die Funktionen add\_paragraph() und add\_run() geben ein Paragraph- bzw. Run-Objekt zurück, damit Sie diese nicht in einem gesonderten Schritt abrufen müssen.

Beachten Sie, dass neue Paragraph-Objekte in Python-Docx zurzeit (Stand: Version 0.5.3) nur am Ende des Dokuments und neue Run-Objekte nur am Ende eines Absatzes hinzugefügt werden können.

Anschließend können Sie erneut die Methode save() aufrufen, um die Änderungen zu speichern.



**Abb. 13–9** Das Dokument nach dem Hinzufügen weiterer Paragraph- und Run-Objekte

Sowohl add\_paragraph() als auch add\_run() akzeptieren als optionales zweites Argument einen String für das Format, das der Absatz bzw. Run aufweisen soll:

```
>>> doc.add_paragraph('Hello world!', 'Title')
```

Durch diese Zeile fügen Sie einen Absatz mit dem Text *Hello world!* und dem Format *Title* hinzu.

## Überschriften hinzufügen

Mit `add_heading()` fügen Sie einen Absatz mit einem Überschriftenformat hinzu:

```
>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.add_heading('Header 1', 1)
<docx.text.Paragraph object at 0x00000000036CB630>
>>> doc.add_heading('Header 2', 2)
<docx.text.Paragraph object at 0x00000000036CB828>
>>> doc.add_heading('Header 3', 3)
<docx.text.Paragraph object at 0x00000000036CB2E8>
>>> doc.add_heading('Header 4', 4)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.save('headings.docx')
```

Die Argumente von `add_heading()` sind ein String mit dem Text der Überschrift und ein Integer von 0 bis 4. Mit 0 erhält die Überschrift das Format *Title*, die für den Titel des Gesamtdokuments verwendet wird. Die Werte 1 bis 4 stehen für die verschiedenen Überschriftenebenen, von 1 für die Hauptüberschrift bis 4 für die niedrigste Ebene von Unterthemen. Außerdem gibt die Funktion `add_heading()` das entsprechende Paragraph-Objekt zurück, sodass Sie es nicht in einem gesonderten Schritt aus dem Document-Objekt abrufen müssen.

Die resultierende Datei *headings.docx* sehen Sie in Abb. 13–10.

## Header 0

---

Header 1  
Header 2  
Header 3  
Header 4

**Abb. 13–10** Das Dokument *headings.docx* mit Überschriftenebenen 0 bis 4

## Zeilenwechsel und Seitenumbrüche hinzufügen

Um einen Zeilenwechsel einzufügen (anstatt einen neuen Absatz zu beginnen), rufen Sie die Methode `add_break()` für das Run-Objekt auf, hinter dem der Wechsel erfolgen soll. Für einen Seitenumbruch übergeben Sie `add_break()` den Wert `docx`.

text.WD\_BREAK.PAGE als einziges Argument. Das können Sie in der Mitte des folgenden Beispiels erkennen:

```
>>> doc = docx.Document()
>>> doc.add_paragraph('This is on the first page!')
<docx.text.Paragraph object at 0x000000003785518>
>>> doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE) ❶
>>> doc.add_paragraph('This is on the second page!')
<docx.text.Paragraph object at 0x0000000037855F8>
>>> doc.save('twoPage.docx')
```

Dadurch wird ein zweiseitiges Word-Dokument mit dem Text *This is on the first page!* auf der ersten und *This is on the second page!* auf der zweiten Seite. Hinter dem Text *This is on the first page!* ist zwar noch reichlich Platz, aber durch den Seitenenumbruch nach dem ersten Run des ersten Absatzes haben wir dafür gesorgt, dass der nächste Absatz trotzdem erst auf der zweiten Seite beginnt (❶).

## Bilder einfügen

Document-Objekte verfügen auch über die Methode add\_picture(), mit der Sie am Ende des Dokuments ein Bild hinzufügen können. Wenn Sie beispielsweise die Datei *zophie.png*, die sich im aktuellen Arbeitsverzeichnis befindet, mit einer Breite von 1 Zoll und einer Höhe von 4 cm (Word kann sowohl mit britischen als auch mit metrischen Einheiten umgehen) am Ende des Dokuments anhängen wollen, geben Sie folgenden Code ein:

```
>>> doc.add_picture('zophie.png', width=docx.shared.Inches(1),
height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x0000000036C7D30>
```

Das erste Argument ist ein String mit dem Dateinamen des Bilds. Mit den optionalen Schlüsselwortargumenten `width` und `height` geben Sie die Breite und Höhe des Bilds im Dokument an. Wenn Sie diese Angaben weglassen, wird das Bild in seiner normalen Größe dargestellt.

Um die Höhe und Breite des Bilds anzugeben, ist es am besten, mit vertrauten Einheiten wie Zoll oder Zentimeter zu arbeiten. Dazu können Sie zur Angabe der Schlüsselwortargumente `width` und `height` auf die Funktionen `docx.shared.Inches()` bzw. `docx.sharedCm()` zurückgreifen.

## Zusammenfassung

Texte sind nicht nur in reinen Textdateien enthalten; wahrscheinlich werden Sie sogar viel öfter mit PDF- oder Word-Dokumenten zu tun haben. Mit dem Modul PyPDF2 können Sie PDF-Dokumente lesen und schreiben. Da das Format PDF ziemlich kompliziert aufgebaut ist, stellt der String, der sich beim Lesen in einem PDF-Dokument ergibt, leider nicht immer eine perfekte Wiedergabe des Textes dar und manche PDFs lassen sich überhaupt nicht lesen. In solchen Fällen haben Sie einfach Pech gehabt und können nur darauf hoffen, dass künftige Versionen von PyPDF2 zusätzliche Möglichkeiten bieten.

Word-Dokumente können Sie mit dem Modul python-docx lesen und hier sind die Ergebnisse viel zuverlässiger. Der Text in Word-Dokumenten lässt sich mithilfe von Paragraph- und Run-Objekten bearbeiten. Diesen Objekten können Sie auch Formate mitgeben, wobei Sie allerdings auf die Standardformate von Word und die Formate beschränkt sind, die sich bereits in dem Dokument befinden. Einem Dokument können Sie außerdem neue Absätze, Überschriften, Umbrüche und Bilder hinzufügen, allerdings nur am Ende.

Die meisten Einschränkungen, auf die Sie bei der Arbeit mit PDF- und Word-Dokumenten stoßen, ergeben sich daraus, dass diese Formate zur ansprechenden Aufbereitung für menschliche Leser gedacht sind und nicht für die einfache Analyse durch Software. Im nächsten Kapitel sehen wir uns zwei weitere weit verbreitete Formulare zur Speicherung von Informationen an, nämlich JSON- und CSV-Dateien. Da sie eigens zur Verwendung durch Computer gedacht sind, kann Python damit viel besser umgehen.

## Wiederholungsfragen

1. Der Funktion `PyPDF2.PdfFileReader()` übergeben Sie *nicht* den Stringwert mit dem Namen der PDF-Datei. Was übergeben Sie stattdessen?
2. In welchen Modi müssen File-Objekte für `PdfFileReader()` und `PdfFileWriter()` geöffnet werden?
3. Wie rufen Sie das `Page`-Objekt für Seite 5 von einem `PdfFileReader`-Objekt ab?
4. In welcher `PdfFileReader`-Variable ist die Anzahl der Seiten in einem PDF-Dokument gespeichert?
5. Was müssen Sie tun, bevor Sie die `Page`-Objekte von einem `PdfFileReader`-Objekt abrufen können, dessen PDF mit dem Passwort `swordfish` geschützt ist?
6. Welche Methoden verwenden Sie, um eine Seite zu drehen?
7. Welche Methode gibt ein `Document`-Objekt für die Datei `demo.docx` zurück?
8. Was ist der Unterschied zwischen einem `Paragraph`- und einem `Run`-Objekt?

9. Wie rufen Sie die Liste der Paragraph-Objekte für ein Document ab, das in der Variablen doc gespeichert ist?
10. Was für Objekte verfügen über die Variablen bold, underline, italic, strike und outline?
11. Was ist der Unterschied zwischen den Werten True, False und None für die Variable bold?
12. Wie erstellen Sie ein Document-Objekt für ein neues Word-Dokument?
13. Wie fügen Sie einem Document-Objekt in der Variablen doc einen Absatz mit dem Text '*Hello there!*' hinzu?
14. Welche Integerwerte können Sie verwenden, um in einem Word-Dokument Überschriftenebenen anzugeben?

## **Übungsprojekte**

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erfüllen:

### **PDF-Paranoia**

Schreiben Sie mit der Funktion `os.walk()` aus Kapitel 9 ein Skript, das alle PDF-Dateien in einem Ordner (und dessen Unterordnern) durchläuft und sie mit dem an der Befehlszeile angegebenen Passwort versieht. Speichern Sie die passwortgeschützten PDFs jeweils mit dem Suffix `_encrypted.pdf` hinter dem ursprünglichen Dateinamen. Bevor das Programm die ursprüngliche Datei löscht, soll es versuchen, die neue Datei zu öffnen, um sicherzustellen, dass kein Fehler unterlaufen ist.

Schreiben Sie anschließend ein Programm, das alle passwortgeschützten PDFs in einem Ordner (und dessen Unterordnern) findet und eine ungeschützte Kopie davon anlegt. Dazu muss das Programm natürlich zunächst das Passwort angeben. Wenn es falsch ist, soll das Programm eine Meldung anzeigen und mit dem nächsten PDF fortfahren.

### **Personalisierte Einladungen als Word-Dokument**

Nehmen wir an, Sie haben eine Textdatei namens `guests.txt`, in der in jeder Zeile der Name eines Partygasts steht:

Prof. Plum  
Miss Scarlet  
Col. Mustard  
Al Sweigart  
RoboCop

Schreiben Sie ein Programm, das ein Word-Dokument mit personalisierten Einladungen wie in Abb. 13–11 erzeugt.

Da Python-Docx nur die Formate verwenden kann, die bereits in dem Word-Dokument vorhanden sind, müssen Sie die gewünschten Formate zunächst in einer leeren Word-Datei anlegen und diese dann mit Python-Docx öffnen. Das fertige Word-Dokument soll eine Einladung pro Seite enthalten. Daher müssen Sie nach dem letzten Absatz einer Einladung immer `add_break()` aufrufen, um einen Zeilenumbruch hinzuzufügen. Dadurch müssen Sie später nur ein einziges Word-Dokument öffnen und können alle Einladungen auf einmal drucken.



Abb. 13–11 Das Word-Dokument mit den personalisierten Einladungen

Ein Beispiel für eine `guests.txt`-Datei können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.

### Brute-Force-Passwortknacker für PDFs

Stellen Sie sich vor, Sie haben das Passwort für ein geschütztes PDF vergessen, wissen aber noch, dass es sich um ein einzelnes englisches Wort gehandelt hat. Das richtige Passwort zu erraten, wäre eine ziemlich langwierige Arbeit. Stattdessen können Sie ein Programm schreiben, das das PDF öffnet, indem es alle möglichen englischen Wörter ausprobiert, bis es das passende gefunden hat. Diese Vorgehensweise wird als *Brute-Force-Angriff* bezeichnet. Laden Sie die Textdatei `dictionary.txt` von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter. Sie enthält mehr als 44.000 englische Wörter, und zwar eines pro Zeile.

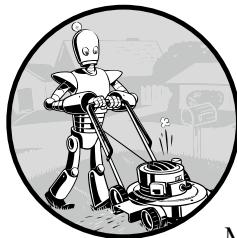
Erstellen Sie mit den Möglichkeiten zum Lesen von Dateien aus Kapitel 8 eine Liste von Wortstrings aus dieser Datei. Durchlaufen Sie dann in einer Schleife alle Wörter in der Liste und übergeben Sie sie jeweils an die Methode `decrypt()`. Wenn diese Methode den Integer 0 zurückgibt, war das Passwort falsch, weshalb das Programm mit dem nächsten Wort weitermachen muss. Gibt `decrypt()` dagegen 1

zurück, soll das Programm die Schleife verlassen und das geknackte Passwort ausgeben. Außerdem sollte das Programm die einzelnen Wörter sowohl komplett in Klein- als auch in Großbuchstaben ausprobieren. (Auf meinem Laptop hat der Durchlauf durch alle 88.000 groß- und kleingeschriebenen Wörter nur wenige Minuten gedauert. Das ist der Grund, aus dem Sie keine einfachen englischen oder deutschen Wörter als Passwörter verwenden dürfen.)



# 14

## Arbeiten mit CSV-Dateien und JSON-Daten



In Kapitel 13 haben Sie gelernt, wie Sie Text aus PDF- und Word-Dokumenten entnehmen. Da diese Dateien in einem Binärformat vorliegen, sind für den Zugriff auf die Daten besondere Python-Module erforderlich. Bei CSV- und JSON-Dateien dagegen handelt es sich um reine Textdateien, die Sie auch in einem Texteditor einsehen können, z. B. im Dateieditor von IDLE. Allerdings hat Python die besonderen Module `csv` und `json`, die jeweils Funktionen aufweisen, um die Arbeit mit diesen Dateiformaten zu vereinfachen.

CSV steht für »comma-separated values«, also »kommagetrennte Werte«. Im Grunde genommen handelt es sich bei CSV-Dateien um vereinfachte Arbeitsblätter, die als Textdateien gespeichert sind. Mit dem Python-Modul `csv` ist das Durchsuchen von CSV-Dateien ganz einfach.

JSON (was gewöhnlich ausgesprochen wird wie der englische Name Jason, also »dschayson«) bedeutet »JavaScript Object Notation« und ist ein Format, mit dem Informationen als JavaScript-Quellcode in Textdateien gespeichert werden können. Um JSON-Dateien zu verwenden, müssen Sie sich in der Programmier-

sprache JavaScript nicht auskennen. Kenntnisse über das Format JSON sind dagegen sehr hilfreich, da es in vielen Webanwendungen eingesetzt wird.

## Das Modul csv

Jede Zeile in einer CSV-Datei steht für eine Zeile in einem Arbeitsblatt, wobei die einzelnen »Zellen« einer Zeile durch Kommata getrennt sind. Beispielsweise sieht das Arbeitsblatt *example.xlsx*, das Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen können, im CSV-Format wie folgt aus:

```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

Diese Datei werde ich in diesem Kapitel in den Beispielen für die interaktive Shell verwenden. Auch *example.csv* können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen. Es ist aber auch möglich, den oben angegebenen Text in einem Texteditor einzugeben und als *example.csv* zu speichern.

CSV-Dateien sind sehr einfach aufgebaut. Viele Merkmale von Excel-Arbeitsblättern sind darin nicht vorhanden, darunter die folgenden:

- Es gibt keine verschiedenen Typen für die Werte – alle sind Strings.
- Es gibt keine Einstellungen für die Schriftgröße oder -farbe.
- Eine Datei kann nicht mehrere Arbeitsblätter enthalten.
- Es ist nicht möglich, die Breite und Höhe der Zellen festzulegen.
- Zellen lassen sich nicht verbinden.
- Es lassen sich keine Bilder oder Diagramme einbetten.

Diese Einfachheit ist aber auch gerade ein Vorteil von CSV-Dateien. Sie können in vielen verschiedenen Arten von Programmen verwendet werden, lassen sich in Texteditoren einsehen (darunter auch im Dateieditor von IDLE) und bilden eine unkomplizierte Möglichkeit, um Daten aus einem Arbeitsblatt darzustellen. Das CSV-Format ist genau das, was der Name besagt: eine reine Textdatei mit Werten, die durch Kommata getrennt sind.

Da CSV-Dateien Textdateien sind, könnten Sie versucht sein, sie als String zu lesen und diesen String dann mit den Techniken aus Kapitel 8 zu verarbeiten. Da die »Zellen« durch Kommata getrennt sind, könnten Sie beispielsweise einfach die Methode `split()` für jede Textzeile aufrufen, um an die Werte zu kommen.

Allerdings muss ein Komma in einer CSV-Datei nicht unbedingt für die Grenze zwischen zwei Zellen stehen, denn für CSV-Dateien gibt es auch Maskierungszeichen, um Kommata und andere Sonderzeichen *als Werte* einzugeben. Die Methode `split()` kann mit solchen Maskierungssequenzen aber nicht umgehen. Aufgrund dieser und anderer möglicher Fallgruben sollten Sie zum Lesen und Schreiben in CSV-Dateien stets das Modul `csv` verwenden.

## Reader-Objekte

Um mit dem Modul `csv` Daten aus einer CSV-Datei zu lesen, müssen Sie ein Reader-Objekt erstellen. Damit können Sie die Zeilen in der Datei durchlaufen. Probieren Sie das wie folgt in der interaktiven Shell aus, wobei sich die Datei `example.csv` im aktuellen Arbeitsverzeichnis befinden muss:

```
>>> import csv ❶
>>> exampleFile = open('example.csv') ❷
>>> exampleReader = csv.reader(exampleFile) ❸
>>> exampleData = list(exampleReader) ❹
>>> exampleData ❺
[['4/5/2015 13:34', 'Apples', '73'], ['4/5/2015 3:41', 'Cherries', '85'],
 ['4/6/2015 12:46', 'Pears', '14'], ['4/8/2015 8:59', 'Oranges', '52'],
 ['4/10/2015 2:07', 'Apples', '152'], ['4/10/2015 18:10', 'Bananas', '23'],
 ['4/10/2015 2:40', 'Strawberries', '98']]
```

Da das Modul `csv` im Lieferumfang von Python enthalten ist, können wir es einfach importieren (❶), ohne es erst installieren zu müssen.

Um mithilfe des Moduls eine CSV-Datei zu öffnen, öffnen Sie diese zunächst wie jede andere Textdatei mit der Funktion `open()` (❷). Anstatt aber die Methoden `read()` oder `readlines()` für das zurückgegebene File-Objekt aufzurufen, übergeben Sie es an die Funktion `csv.reader()` (❸), die ein Reader-Objekt zurückgibt. Beachten Sie, dass Sie der Funktion `csv.reader()` nicht direkt einen String mit dem Dateinamen übergeben können.

Der unmittelbarste Weg, um auf die Werte in dem Reader-Objekt zuzugreifen, besteht darin, es an `list()` zu übergeben, um es in eine einfache Python-Liste umzuwandeln (❹). Die zurückgegebene Liste aus Listen können Sie dann in einer Variablen wie `exampleData` speichern. Wenn Sie nun `exampleData` in der Shell eingeben, wird die Liste angezeigt (❺).

Da Ihnen der Inhalt der CSV-Datei nun als Liste von Listen vorliegt, können Sie mit dem Ausdruck `exampleData[row][col]` auf die Werte in bestimmten Zeilen und Spalten zurückgreifen, wobei `row` der Index für eine der Listen in `exampleData` ist und `col` der Index des gewünschten Elements in dieser Liste. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> exampleData[0][0]
'4/5/2015 13:34'
>>> exampleData[0][1]
'Apples'
>>> exampleData[0][2]
'73'
>>> exampleData[1][1]
'Cherries'
>>> exampleData[6][1]
'Strawberries'
```

exampleData[0][0] sucht die erste Liste auf und gibt den ersten String darin zurück, exampleData[0][2] gibt den dritten String der ersten Liste zurück usw.

### Daten in einer for-Schleife aus Reader-Objekten lesen

Bei umfangreicheren CSV-Dateien sollten Sie das Reader-Objekt in einer for-Schleife durchlaufen, damit Sie nicht die gesamte Datei auf einmal in den Arbeitsspeicher laden müssen:

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
    print('Row #' + str(exampleReader.line_num) + ' ' + str(row))

Row #1 ['4/5/2015 13:34', 'Apples', '73']
Row #2 ['4/5/2015 3:41', 'Cherries', '85']
Row #3 ['4/6/2015 12:46', 'Pears', '14']
Row #4 ['4/8/2015 8:59', 'Oranges', '52']
Row #5 ['4/10/2015 2:07', 'Apples', '152']
Row #6 ['4/10/2015 18:10', 'Bananas', '23']
Row #7 ['4/10/2015 2:40', 'Strawberries', '98']
```

Nachdem Sie das Modul `csv` importiert und aus der CSV-Datei ein Reader-Objekt erstellt haben, durchlaufen Sie dessen Zeilen in einer Schleife. Jede Zeile ist eine Liste von Werten, wobei jeder Wert für eine Zelle steht.

Die Funktion `print()` gibt die Nummer der aktuellen Zeile und deren Inhalt aus. Die Zeilenummer rufen Sie dabei aus der Variablen `line_num` des Reader-Objekts ab.

Ein Reader-Objekt können Sie immer nur einmal durchlaufen. Um die CSV-Datei erneut zu lesen, müssen Sie `csv.reader` abermals aufrufen, um ein neues Reader-Objekt zu erstellen.

## Writer-Objekte

Mit einem Writer-Objekt können Sie Daten in eine CSV-Datei schreiben. Um ein solches Objekt zu erstellen, verwenden Sie die Funktion `csv.writer()`:

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='')    ❶
>>> outputWriter = csv.writer(outputFile)    ❷
>>> outputWriter.writerow(['spam', 'eggs', 'bacon', 'ham'])
21
>>> outputWriter.writerow(['Hello, world!', 'eggs', 'bacon', 'ham'])
32
>>> outputWriter.writerow([1, 2, 3.141592, 4])
16
>>> outputFile.close()
```

Als Erstes wird die Funktion `open()` mit dem Argument `'w'` aufgerufen, um eine Datei im Schreibmodus zu öffnen (❶). Dadurch wird ein Objekt erstellt, das Sie an `csv.writer()` übergeben können (❷), um ein Writer-Objekt anzulegen.

Unter Windows müssen Sie zusätzlich einen leeren String als Schlüsselwortargument `newline` von `open()` übergeben. Aus technischen Gründen, die in diesem Buch nicht erklärt werden können, werden die Zeilen in `output.csv` sonst mit doppeltem Zeilenabstand geschrieben (siehe Abb. 14–1).

	A1	B1	C1	D1	E1	F1	G1
1	42	2	3	4	5	6	7
2							
3	2	4	6	8	10	12	14
4							
5	3	6	9	12	15	18	21
6							
7	4	8	12	16	20	24	28
8							
9	5	10	15	20	25	30	35
10							

**Abb. 14–1** Wenn Sie in `open()` das Schlüsselwortargument `newline=' '` vergessen, bekommt die CSV-Datei einen doppelten Zeilenabstand.

Die Methode `writerow()` für Writer-Objekte nimmt eine Liste als Argument entgegen. Jeder Wert dieser Liste steht in der CSV-Ausgabedatei in einer eigenen Zelle. Der Rückgabewert von `writerow()` ist die Anzahl der Zeichen, die für diese Zeile in die Datei geschrieben wurden (einschließlich Zeilenumbruchzeichen).

Der weiter oben angegebene Beispielcode erstellt die Datei `output.csv`, die wie folgt aussieht:

```
spam,eggs,bacon,ham
>Hello, world!",eggs,bacon,ham
1,2,3.141592,4
```

Das `Writer`-Objekt hat das Komma im Wert 'Hello, world' automatisch maskiert, indem es den String in der CSV-Datei in doppelte Anführungszeichen gesetzt hat. Die Handhabung solcher Sonderfälle nimmt Ihnen das Modul `csv` ab.

## Die Schlüsselwortargumente `delimiter` und `lineterminator`

Nehmen wir an, Sie möchten die Zellen statt durch Kommata durch Tabulatoren trennen und außerdem einen doppelten Zeilenabstand für die Zeilen vorsehen. Das können Sie in der interaktiven Shell wie folgt erreichen:

```
>>> import csv
>>> csvFile = open('example.tsv', 'w', newline='')
>>> csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')
❶
>>> csvWriter.writerow(['apples', 'oranges', 'grapes'])
24
>>> csvWriter.writerow(['eggs', 'bacon', 'ham'])
17
>>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam', 'spam'])
32
>>> csvFile.close()
```

Dadurch werden das Trennzeichen und das Zeilenendezeichen in der Datei geändert. Das *Trennzeichen (delimiter)* ist das Zeichen, das zwischen den Zellen einer Zeile steht. Standardmäßig wird in CSV-Dateien dafür ein Komma verwendet. Das *Zeilenendezeichen (line terminator)* ist das Zeichen, das das Ende einer Zeile markiert, standardmäßig das Zeilenumbruchzeichen. Mit den Schlüsselwortargumenten `delimiter` und `lineterminator` von `csv.writerow()` können Sie andere Werte für diese besonderen Zeichen festlegen.

Wenn Sie `delimiter='\t'` und `lineterminator='\n\n'` übergeben (❶), wird als Trennzeichen zwischen den Zellen einer Zeile ein Tabulator verwendet und als Zeilenendezeichen ein doppelter Zeilenumbruch.

Um drei Zeilen zu schreiben, rufen wir `writerow()` dreimal auf. Dadurch entsteht die Datei `example.tsv` mit folgendem Inhalt:

```
apples    oranges    grapes
eggs      bacon      ham
spam       spam       spam       spam       spam
```

Da die Zellen jetzt durch Tabulatoren getrennt sind, verwenden wir die Dateierung `.tsv` für »tab-separated values«.

## Projekt: Kopfzeilen aus CSV-Dateien entfernen

Stellen Sie sich vor, Sie habe die langweilige Aufgabe, jeweils die erste Zeile aus mehreren hundert CSV-Dateien zu entfernen, etwa weil die Dateien anschließend einem automatischen Vorgang zugeführt werden, für den nur die Daten und nicht die Kopfzeile mit den Spaltenbezeichnungen gebraucht wird. Sie *könnten* jede Datei in Excel öffnen, die erste Zeile löschen und die Datei speichern, aber das würde Stunden dauern. Stattdessen wollen wir ein Programm dafür schreiben.

Dieses Programm muss alle Dateien mit der Endung `.csv` im aktuellen Arbeitsverzeichnis öffnen, den Inhalt einlesen und ohne die erste Zeile in eine Datei mit demselben Namen schreiben. Dadurch werden die alten CSV-Dateien durch neue Versionen mit demselben Inhalt, aber ohne die Kopfzeilen ersetzt.

### Warnung

Wenn Sie ein Programm schreiben, das Dateien bearbeitet, sollten Sie die Dateien zuvor sichern, nur für den Fall, dass das Programm nicht wie erwartet funktioniert. Es wäre schlecht, wenn Sie die Originaldateien versehentlich löschen würden.

Das Programm muss folgende Aufgaben erfüllen:

- Alle CSV-Dateien im aktuellen Arbeitsverzeichnis finden
- Den kompletten Inhalt jeder Datei einlesen
- Den Inhalt ohne die erste Zeile in eine neue CSV-Datei schreiben

Dazu muss der Code Folgendes erledigen:

- Die von `os.listdir()` zurückgegebene Liste der Dateien durchlaufen und dabei alle Nicht-CSV-Dateien überspringen
- Ein Reader-Objekt erstellen, den Inhalt der Datei lesen und dabei anhand des Attributs `line_num` die zu überspringende Zeile bestimmen
- Ein Writer-Objekt erstellen und die eingelesenen Daten in die neue Datei schreiben

Öffnen Sie ein neues Dateieditorfenster für dieses Projekt und speichern Sie es als `removeCsvHeader.py`.

### Schritt 1: Alle CSV-Dateien durchlaufen

Als Erstes muss das Programm alle CSV-Dateien im aktuellen Arbeitsverzeichnis durchlaufen. Geben Sie in `removeCsvHeader.py` folgenden Code ein:

```

#! python3
# removeCsvHeader.py - Entfernt die Kopfzeilen aus allen CSV-Dateien im
# aktuellen Arbeitsverzeichnis

import csv, os

os.makedirs('headerRemoved', exist_ok=True)

# Durchläuft alle Dateien im aktuellen Arbeitsverzeichnis
for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
        continue # Überspringt Nicht-CSV-Dateien ❶

    print('Removing header from ' + csvFilename + '...')

    # TODO: CSV-Datei einlesen (und erste Zeile überspringen)

    # TODO: CSV-Datei schreiben

```

Die Funktion `os.makedirs()` erstellt den Ordner *headerRemoved*, in den die CSV-Dateien ohne Kopfzeilen geschrieben werden sollen. Mit einer `for`-Schleife über `os.listdir('.')` kommen wir unserem Ziel schon sehr nahe, doch da sie *alle* Dateien im Arbeitsverzeichnis durchläuft, müssen wir noch etwas Code hinzufügen, um Dateien zu überspringen, deren Namen nicht auf *.csv* enden. Die Anweisung `continue` (❶) sorgt dafür, dass die `for`-Schleife bei einer Nicht-CSV-Datei zum nächsten Dateinamen übergeht.

Damit es bei der Ausführung des Programms überhaupt irgendeine Ausgabe gibt, lassen Sie die Meldung anzeigen, an welcher CSV-Datei es gerade arbeitet. Die `TODO`-Kommentare beschreiben, was in dem Rest des Programms noch zu erledigen ist.

## Schritt 2: Die CSV-Datei lesen

Dieses Programm entfernt nicht die erste Datei aus der CSV-Datei, sondern erstellt eine Kopie dieser Datei ohne die erste Zeile. Da der Dateiname der Kopie mit dem Namen der ursprünglichen Datei überschrieben ist, wird das Original überschrieben.

Das Programm muss erkennen können, ob es in der Schleife gerade die erste Zeile anfasst. Fügen Sie dazu folgenden Code zu *removeCsvHeader.py* hinzu:

```

#! python3
# removeCsvHeader.py - Entfernt die Kopfzeilen aus allen CSV-Dateien im
# aktuellen Arbeitsverzeichnis

-- schnipp --

```

```
# Liest die CSV-Datei (und überspringt die erste Zeile)
csvRows = []
csvFileObj = open(csvFilename)
readerObj = csv.reader(csvFileObj)
for row in readerObj:
    if readerObj.line_num == 1:
        continue # Überspringt die erste Zeile
    csvRows.append(row)
csvFileObj.close()

# TODO: CSV-Datei schreiben
```

Anhand des Attributs `line_num` des Reader-Objekts können Sie erkennen, welche Zeile der CSV-Datei gerade gelesen wird. In einer weiteren `for`-Schleife durchlaufen Sie die von dem Reader-Objekt zurückgegebenen Zeilen, woraufhin dann alle außer der ersten an `csvRows` angehängt werden.

Während die `for`-Schleife Zeile für Zeile durchgeht, prüft der Code, ob `readerObj.line_num` gleich 1 ist. Wenn ja, wird `continue` ausgeführt, um zur nächsten Zeile vorzurücken, ohne die aktuelle Zeile an `csvRows` anzuhängen. Bei allen nachfolgenden Zeilen ist die Bedingung stets `False`, weshalb alle in `csvRows` aufgenommen werden.

### Schritt 3: Die CSV-Datei ohne die erste Zeile schreiben

Die Liste `csvRows`, die am Ende alle Zeile außer der ersten enthält, muss schließlich in eine CSV-Datei im Ordner `headerRemoved` geschrieben werden. Ergänzen Sie:

```
#! python3
# removeCsvHeader.py - Entfernt die Kopfzeilen aus allen CSV-Dateien im
# aktuellen Arbeitsverzeichnis

-- schnipp --

# Durchläuft alle Dateien im aktuellen Arbeitsverzeichnis
for csvFilename in os.listdir('.'): ❶
    if not csvFilename.endswith('.csv'):
        continue # Überspringt Nicht-CSV-Dateien

-- schnipp --

# Schreibt die CSV-Datei
csvFileObj = open(os.path.join('headerRemoved', csvFilename), 'w',
                  newline='')
csvWriter = csv.writer(csvFileObj)
for row in csvRows:
    csvWriter.writerow(row)
csvFileObj.close()
```

Das Writer-Objekt schreibt die Liste in eine CSV-Datei in `headerRemoved`. Als Dateiname wird dabei der Wert der `csvFilename` genommen (die auch schon vom Reader-Objekt verwendet wurde). Dadurch wird die ursprüngliche Datei überschrieben.

Nachdem wir das Writer-Objekt erstellt haben, durchlaufen wir die in `csvRows` gespeicherten Unterlisten und schreiben sie jeweils in die Datei.

Wenn dieser Code ausgeführt ist, geht die äußere for-Schleife (❶) zur nächsten Datei in `os.listdir('.')` über. Nachdem auch diese Schleife komplett durchgelaufen ist, ist das Programm beendet.

Um das Programm zu testen, laden Sie *removeCsvHeader.zip* von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter. Entpacken Sie den Ordner und führen Sie *removeCsvHeader.py* darin aus. Dabei erhalten Sie folgende Ausgabe:

```
Removing header from NAICS_data_1048.csv...
Removing header from NAICS_data_1218.csv...
-- schnipp --
Removing header from NAICS_data_9834.csv...
Removing header from NAICS_data_9986.csv...
```

Jedes Mal, wenn das Programm die erste Zeile einer CSV-Datei entfernt, gibt es den Namen der betreffenden Datei aus.

## Vorschläge für ähnliche Programme

Die Programme, die Sie zur Verarbeitung von CSV-Dateien schreiben können, ähneln denen für Excel-Dateien, da beide Arten von Dateien Arbeitsblätter enthalten. Mit solchen Programmen lassen sich beispielsweise folgende Aufgaben lösen:

- Daten in unterschiedlichen Zeilen einer CSV-Datei oder in unterschiedlichen CSV-Dateien vergleichen
- Daten von einer CSV- in eine Excel-Datei kopieren oder umgekehrt
- CSV-Dateien auf ungültige Daten oder falsche Formatierungen überprüfen und solche Fehler melden
- Daten aus einer CSV-Datei als Eingabe für ein Python-Programm lesen

## JSON und APIs

JSON (Java Script Object Notation) ist eine häufig genutzte Möglichkeit, um Daten in Form eines einzelnen, für Menschen lesbaren Strings darzustellen. Es handelt sich dabei um die Art und Weise, in der JavaScript-Programme Datenstrukturen

schreiben. Sie ähnelt darüber hinaus auch dem, was die Python-Funktion `pprint()` ausgibt. Um mit JSON-Daten arbeiten zu können, sind Kenntnisse in JavaScript jedoch nicht nötig.

Das folgende Beispiel zeigt, wie Daten im JSON-Format aussehen:

```
{"name": "Zophie", "isCat": true,  
"miceCaught": 0, "napsTaken": 37.5,  
"felineIQ": null}
```

Es ist gut, sich mit JSON auszukennen, da viele Websites JSON-Inhalte als eine Möglichkeit bereitstellen, über die Programme mit der Website arbeiten können. Eine solche Einrichtung wird als *Anwendungsprogrammierschnittstelle* (Application Programming Interface, API) bezeichnet. Der Zugriff auf eine API ist mit dem Zugriff auf eine andere Webseite über deren URL vergleichbar, wobei der Unterschied jedoch darin besteht, dass die von einer API zurückgegebenen Daten für die Verwendung durch Computer formatiert sind (z. B. in JSON). Für Menschen sind APIs nur schwer zugänglich.

Viele Websites stellen ihre Daten im JSON-Format zur Verfügung. Facebook, Twitter, Yahoo, Google, Tumblr, Wikipedia, Flickr, Data.gov, Reddit, IMDb, Rotten Tomatoes, LinkedIn und viele andere beliebte Websites bieten APIs für die Nutzung durch Programme an. Bei manchen dieser Websites ist eine Registrierung erforderlich, die jedoch meistens kostenlos ist. Um die gewünschten Daten abrufen zu können, müssen Sie in der zugehörigen Dokumentation nachsehen, welche URLs Ihr Programm anfordern muss und welches allgemeine Format die zurückgegebenen JSON-Datenstrukturen aufweisen. Eine solche Dokumentation wird von der Website mit der API bereitgestellt. Wenn es auf dieser Website eine Seite für Entwickler gibt (*Developer*), sollten Sie dort nach der Dokumentation Ausschau halten.

Mithilfe von APIs können Sie Programme schreiben, die Folgendes tun:

- Rohdaten von Websites abschöpfen (Der Zugriff über eine API ist häufig komfortabler, als die Webseiten herunterzuladen und den HTML-Text mit Beautiful Soup zu durchforsten.)
- Neue Posts von einem Ihrer Social-Network-Konten automatisch herunterladen und in einem anderen Konto einstellen. Damit können Sie beispielsweise Ihre Tumblr-Posts in Facebook wiederholen.
- Eine »Filmenzyklopädie« für Ihre private Filmsammlung zusammenstellen, indem Sie Daten von IMDb, Rotten Tomatoes und Wikipedia abrufen und auf Ihrem Computer zu einer einzigen Textdatei kombinieren.

In den Ressourcen auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) finden Sie Beispiele für JSON-APIs.

## Das Modul json

Das Python-Modul `json` nimmt Ihnen mit den Funktionen `json.loads()` und `json.dumps()` die Einzelheiten der Übersetzung zwischen Strings mit JSON-Daten und Python-Werten ab. Nicht alle Python-Werte können im JSON-Format gespeichert werden, sondern nur Strings, Integer, Fließkommazahlen, boolesche Werte, Listen, Dictionary und Werte vom Datentyp `None`. Python-spezifische Objekte wie File-Objekte, Reader- und Writer-Objekte für CSV-Dateien, Regex-Objekte oder WebElement-Objekte von Selenium lassen sich durch JSON nicht darstellen.

### JSON-Daten mit der Funktion `loads()` laden

Um einen String mit JSON-Daten in einen Python-Wert zu übersetzen, übergeben Sie ihn der Funktion `json.loads()`. (Der Name der Funktion bedeutet übrigens nicht *loads*, also »lädt«, sondern ist eine Abkürzung für *load string*, also »String laden«.) Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0,
   "felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

Nach dem Import des Moduls `json` können Sie die Funktion `loads()` aufrufen und ihr einen String mit JSON-Daten übergeben. Beachten Sie, dass JSON-Strings immer in doppelten Anführungszeichen stehen. Die Funktion gibt die Daten anschließend in Form eines Python-Dictionarys zurück. Da Python-Dictionarys nicht geordnet sind, können die Schlüssel-Wert-Paare darin in einer anderen Reihenfolge ausgegeben werden als in dem ursprünglichen JSON-String.

### JSON-Daten mit der Funktion `dumps()` schreiben

Die Funktion `json.dumps()` (auch hier bedeutet der Name wiederum *dump string*) übersetzt einen Python-Wert in einen JSON-Datenstring:

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie',
   'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```

Als Werte sind nur die grundlegenden Python-Datentypen Dictionary, Liste, Integer, Fließkommazahl, String, boolescher Wert und None zulässig.

## Projekt: Die aktuellen Wetterdaten abrufen

Die Wettervorhersage abzurufen, scheint eine ganz einfache Aufgabe zu sein: Sie öffnen Ihren Browser, klicken in die Adressleiste, geben den URL einer entsprechenden Website ein (oder suchen danach und klicken dann auf den Link), warten bis die Seite geladen ist, machen die Wetterinformationen zwischen den ganzen Werbeeinblendungen ausfindig usw.

Tatsächlich gibt es eine ganze Menge langweiliger Schritte, die Sie sich sparen können, wenn Sie ein Programm haben, das die Wettervorhersage für die nächsten Tage herunterlädt und in Textform ausgibt. Zum Herunterladen der Daten aus dem Web kann ein solches Programm das Modul `requests` aus Kapitel 11 verwenden.

Das Programm führt also folgende Schritte aus:

- Den Standort von der Befehlszeile abrufen
- Die JSON-Wetterdaten von *OpenWeatherMap.org* herunterladen
- Den JSON-Datenstring in eine Python-Datenstruktur umwandeln
- Das Wetter für heute, morgen und übermorgen ausgeben

Der Code muss dazu Folgendes tun:

- Die Strings in `sys.argv` verknüpfen, um den Standort zu erhalten
- Die Wetterdaten mit `requests.get()` herunterladen
- Die JSON-Daten mit `json.loads()` in eine Python-Datenstruktur umwandeln
- Die Wettervorhersage ausgeben

Öffnen Sie für dieses Projekt ein neues Dateieditorfenster und speichern Sie es als *quickWeather.py*.

### Schritt 1: Den Standort aus dem Befehlszeilenargument entnehmen

Die Eingabe für das Programm kommt von der Befehlszeile. Schreiben Sie folgenden Code:

```
#! python3
# quickWeather.py - Gibt die Wettervorhersage für den an der Befehlszeile
# eingegebenen Standort aus

import json, requests, sys
```

```

# Ermittelt den Standort aus den Befehlszeilenargumenten
if len(sys.argv) < 2:
    print('Usage: quickWeather.py location')
    sys.exit()
location = ' '.join(sys.argv[1:])

# TODO: JSON-Daten von OpenWeatherMap.org herunterladen

# TODO: JSON-Daten in eine Python-Variablen laden

```

Die Befehlszeilenargumente werden in Python in der Liste `sys.argv` gespeichert. Nach der Shebang-Zeile (`#!`) und den `import`-Anweisungen prüft das Programm, ob es mehr als ein Befehlszeilenargument gibt. (Wie Sie bereits wissen, enthält `sys.argv` immer mindestens ein Element, nämlich `sys.argv[0]` mit dem Dateinamen des Python-Skripts.) Enthält die Liste nur ein einziges Element, dann hat der Benutzer keinen Standort angegeben, weshalb dem Benutzer eine Meldung über die richtige Nutzung des Programms angezeigt wird, bevor das Programm endet.

Da zur Trennung von Befehlszeilenargumenten Leerzeichen verwendet werden, wird bei einer Angabe wie San Francisco, CA der Wert `['quickWeather.py', 'San', 'Francisco', 'CA']` in `sys.argv` gespeichert. Um an den richtigen Standort zu kommen, müssen Sie daher alle Strings außer dem ersten mit der Methode `join()` verketten. Anschließend wird der Gesamtstring in der Variable `location` gespeichert.

## Schritt 2: Die JSON-Daten herunterladen

*OpenWeatherMap.org* stellt Echtzeit-Wetterinformationen im JSON-Format bereit. Ihr Programm muss einfach nur die Seite `http://api.openweathermap.org/data/2.5/forecast/daily?q=<Standort>&cnt=3` herunterladen, wobei `<Standort>` der Name der gewünschten Stadt ist. Ergänzen Sie `quickWeather.py` wie folgt:

```

#!/usr/bin/python3
# quickWeather.py - Gibt die Wettervorhersage für den an der Befehlszeile
# eingegebenen Standort aus

-- schnipp --

# Lädt die JSON-Daten von OpenWeatherMap.org herunter
url = 'http://api.openweathermap.org/data/2.5/forecast/daily?q=%s&cnt=3' %
      (location)
response = requests.get(url)
response.raise_for_status()
# TODO: JSON-Daten in eine Python-Variablen laden

```

Den Wert von `location` haben wir aus den Befehlszeilenargumenten bezogen. Um den gewünschten URL-String zusammenzustellen, verwenden wir den Platz-

halter %s und fügen an dieser Stelle den String ein, der sich in location befindet. Das Ergebnis speichern wir in der Variablen url, die wir dann an requests.get() übergeben. Diese Funktion gibt ein Response-Objekt zurück, das Sie mit raise\_for\_status() auf Fehler überprüfen können. Wenn keine Ausnahme ausgelöst wurde, befindet sich der heruntergeladene Text in response.text.

### Schritt 3: JSON-Daten laden und die Wettervorhersage ausgeben

Die Membervariable response.text enthält einen umfangreichen String mit JSON-Daten. Um ihn in einen Python-Wert umzuwandeln, rufen Sie die Funktion json.loads() auf. Die JSON-Daten sehen wie folgt aus:

```
{'city': {'coord': {'lat': 37.7771, 'lon': -122.42},
           'country': 'United States of America',
           'id': 5391959,
           'name': 'San Francisco',
           'population': 0},
  'cnt': 3,
  'cod': '200',
  'list': [{"clouds": 0,
            'deg': 233,
            'dt': 1402344000,
            'humidity': 58,
            'pressure': 1012.23,
            'speed': 1.96,
            'temp': {'day': 302.29,
                      'eve': 296.46,
                      'max': 302.29,
                      'min': 289.77,
                      'morn': 294.59,
                      'night': 289.77},
            'weather': [{"description": "sky is clear",
                         'icon': '01d'},
                        ...
                        -- schnipp --
```

Diese Daten können Sie einsehen, wenn Sie weatherData an pprint.pprint() übergeben. Was die einzelnen Felder bedeuten, können Sie in der Onlinedokumentation auf <http://openweathermap.org/> herausfinden. Beispielsweise ist die Angabe 302.29 hinter 'day' die Tagstemperatur in Kelvin, nicht Celsius oder Fahrenheit.

Die Wetterangaben, an denen wir interessiert sind, befinden sich in 'main' und 'description'. Um sie sauber auszugeben, ergänzen Sie *quickWeather.py* wie folgt:

```
#! python3
# quickWeather.py - Gibt die Wettervorhersage für den an der Befehlszeile
# eingegebenen Standort aus
...
-- schnipp --
```

```
# Lädt die JSON-Daten in eine Python-Variable
weatherData = json.loads(response.text)
# Gibt die Wettervorhersage aus
w = weatherData['list'] ❶
print('Current weather in %s:' % (location))
print(w[0]['weather'][0]['main'], '-', w[0]['weather'][0]['description'])
print()
print('Tomorrow:')
print(w[1]['weather'][0]['main'], '-', w[1]['weather'][0]['description'])
print()
print('Day after tomorrow:')
print(w[2]['weather'][0]['main'], '-', w[2]['weather'][0]['description'])
```

Im Code wird `weatherData['list']` in der Variablen `w` gespeichert, um Ihnen ein wenig Tipparbeit abzunehmen (❶), denn anschließend können Sie einfach `w[0]`, `w[1]` und `w[2]` verwenden, um die Dictionarys mit den Wettervorhersagen für heute, morgen und übermorgen abzurufen. Jedes dieser Dictionarys verfügt über den Schlüssel `'weather'`, dessen Wert eine Liste ist. Das erste Listenelement (am Index 0) ist ein verschachteltes Dictionary mit weiteren Schlüsseln. In diesem Programm geben wir die Werte aus, die unter den Schlüsseln `'main'` und `'description'` gespeichert sind. Dabei trennen wir sie durch einen Bindestrich.

Wenn Sie dieses Programm als `quickWeather.py` San Francisco, CA an der Befehlszeile aufrufen, erhalten Sie folgende Ausgabe:

```
Current weather in San Francisco, CA:
Clear - sky is clear

Tomorrow:
Clouds - few clouds

Day after tomorrow:
Clear - sky is clear
```

(Das Wetter ist einer der Gründe, warum ich gern in San Francisco lebe!)

## Vorschläge für ähnliche Programme

Der Zugriff auf Wetterdaten kann die Grundlage für viele verschiedene Programme sein. Damit können Sie folgende Aufgaben erledigen:

- Wettervorhersagen für mehrere Campingplätze und Wanderrouten sammeln, um zu sehen, wo das beste Wetter zu erwarten ist

- Die Wettervorhersage regelmäßig nach einem festen Zeitplan prüfen und eine E-Mail-Benachrichtigung mit einer Frostwarnung senden, damit Sie Ihre Pflanzen ins Haus holen (Zeitpläne werden in Kapitel 15 erklärt, das Senden von E-Mails in Kapitel 16)
- Wetterdaten von mehreren Websites abrufen, um alle auf einmal anzuzeigen oder um Durchschnittswerte der verschiedenen Vorhersagen zu berechnen und auszugeben

## Zusammenfassung

CSV und JSON sind weit verbreitete Textformate zur Datenspeicherung. Sie lassen sich von Programmen sehr leicht analysieren und sind trotzdem noch für Menschen lesbar, sodass sie oft für einfache Arbeitsblätter oder für die Daten von Webanwendungen genutzt werden. Die Module `csv` und `json` vereinfachen das Lesen und Schreiben von CSV- bzw. JSON-Dateien.

In den letzten Kapiteln haben Sie erfahren, wie Sie mithilfe von Python Informationen aus vielen verschiedenen Dateiformaten entnehmen können. Eine häufige Aufgabe besteht darin, Daten zu entnehmen und nach bestimmten Informationen zu durchsuchen. Die Umstände sind dabei oft so spezifisch, dass kommerzielle Software nicht die optimale Hilfe bietet. Mit selbst geschriebenen Skripten können Sie jedoch große Mengen von Daten in den besprochenen Formaten handhaben.

In Kapitel 15 verlassen wir das Thema Datenformate und beschäftigen uns damit, wie wir Aufgaben automatisch nach einem Zeitplan ausführen lassen können.

## Wiederholungsfragen

1. Welche Merkmale von Excel-Arbeitsblättern weisen CSV-Dateien nicht auf?
2. Was übergeben Sie `csv.reader()` und `csv.writer()`, um ein Reader- bzw. Writer-Objekt zu erstellen?
3. In welchem Modus müssen File-Objekte für Reader- und Writer-Objekte geöffnet werden?
4. Welche Methode nimmt ein Listenargument entgegen und schreibt es in eine CSV-Datei?
5. Was bewirken die Schlüsselwortargumente `delimiter` und `lineterminator`?
6. Welche Funktion nimmt einen JSON-Datenstring entgegen und gibt eine Python-Datenstruktur zurück?
7. Welche Funktion nimmt eine Python-Datenstruktur entgegen und gibt einen JSON-Datenstring zurück?

## Übungsprojekt

Schreiben Sie zur Übung ein Projekt, das die folgende Aufgabe erfüllt:

### Excel-in-CSV-Konverter

In Excel können Sie ein Arbeitsblatt mit nur wenigen Mausklicks als CSV-Datei speichern. Wenn Sie das jedoch für Hunderte von Excel-Dateien tun müssten, kann das stundenlange Klickarbeit bedeuten. Schreiben Sie mithilfe des Moduls `openpyxl` aus Kapitel 12 ein Programm, das alle Excel-Dateien im aktuellen Arbeitsverzeichnis liest und als CSV-Dateien ausgibt.

Da eine Excel-Datei mehrere Arbeitsblätter enthalten kann, müssen Sie eine CSV-Datei pro *Arbeitsblatt* erstellen. Die Namen der CSV-Dateien sollen dem Muster `<Excel-Dateiname>_<Arbeitsblattnname>.csv` folgen, wobei mit dem Excel-Dateinamen der Name ohne die Erweiterung gemeint ist (also `spam_data`, nicht `spam_data.xlsx`). Den Namen des Arbeitsblatts finden Sie in der Variablen `title` des `Worksheet`-Objekts.

Dieses Programm enthält viele verschachtelte `for`-Schleifen. Das Grundgerüst sieht wie folgt aus:

```
for excelFile in os.listdir('.'):
    # Überspringt Nicht-XLSX-Dateien, lädt das Workbook-Objekt
    for sheetName in wb.get_sheet_names():
        # Durchläuft alle Arbeitsblätter in der Arbeitsmappe
        sheet = wb.get_sheet_by_name(sheetName)

        # Stellt den CSV-Dateinamen aus dem Excel-Dateinamen und dem Namen des
        # Arbeitsblatts zusammen
        # Erstellt das csv.writer-Objekt für die CSV-Datei

        # Durchläuft alle Zeilen im Arbeitsblatt
        for rowNum in range(1, sheet.get_highest_row() + 1):
            rowData = [] # Hängt jede Zelle an diese Liste an
            # Durchläuft alle Zellen in der Zeile
            for colNum in range(1, sheet.get_highest_column() + 1):
                # Hängt die Daten jeder Zelle an rowData an

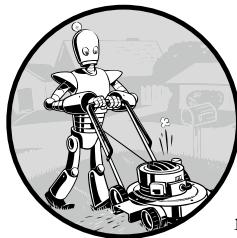
                # Schreibt die Liste rowData in die CSV-Datei

    csvFile.close()
```

Laden Sie die ZIP-Datei `excelSpreadsheets.zip` von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunter und entpacken Sie sie in dem Verzeichnis, in dem sich auch das Programm befindet. Diese Dateien können Sie zum Testen des Programms verwenden.

# 15

## Zeit einhalten, Aufgaben zeitlich planen und Programme starten



Es ist zwar gut und schön, Programme ausführen zu können, während Sie selbst am Computer sitzen, aber viel praktischer wäre es, wenn ein Programm auch unbeaufsichtigt laufen könnte. Mithilfe der Systemuhr des Computers können Sie dafür sorgen, dass Code zu einer bestimmten Uhrzeit an einem bestimmten Datum oder in regelmäßigen Abständen ausgeführt wird. Beispielsweise können Sie ein Programm schreiben, das stündlich nach neuen Informationen auf einer Website sucht und gegebenenfalls herunterlädt, oder eines, das eine rechenintensive Aufgabe nachts um 4 Uhr ausführt, während Sie den Computer ohnehin nicht anderweitig brauchen. Mit den Python-Modulen `time` und `datetime` ist all dies möglich.

Mit den Modulen `subprocess` und `threading` können Sie auch Programme schreiben, die andere Programme nach einem Zeitplan starten. Der schnellste Weg der Programmierung besteht oft darin, Anwendungen zu nutzen, die andere Leute schon geschrieben haben.

## Das Modul time

Die Systemuhr Ihres Computers ist auf ein bestimmtes Datum, eine bestimmte Uhrzeit und eine bestimmte Zeitzone eingestellt. Mithilfe des integrierten Moduls `time` können Python-Programme die aktuelle Uhrzeit der Systemuhr ablesen. Besonders nützlich sind die Funktionen `time.time()` und `time.sleep()` dieses Moduls.

### Die Funktion `time.time()`

Die *Unix-Epoche* ist ein Referenzzeitpunkt, der in der Programmierung häufig verwendet wird, nämlich 0 Uhr am 1. Januar 1970 UTC (Coordinated Universal Time; französisch: Temps Universel Coordonné). Die Funktion `time.time()` gibt die Anzahl der seit diesem Zeitpunkt verstrichenen Sekunden als Fließkommawert zurück. (Ein Fließkommawert ist einfach nur eine Zahl mit einem Dezimalpunkt.) Diese Zahl wird auch als *Epochen-Zeitstempel* bezeichnet. Das können Sie wie folgt in der interaktiven Shell ausprobieren:

```
>>> import time  
>>> time.time()  
1425063955.068649
```

Für dieses Beispiel habe ich `time.time()` am 27. Februar 2015 um 11.05 Uhr Pacific Standard Time oder 19.05 Uhr UTC aufgerufen. Der Rückgabewert gibt an, wie viele Sekunden zwischen der Unix-Epoche und dem Aufruf von `time.time()` vergangen sind.

### Hinweis

In den Beispielen für die interaktive Shell sehen Sie die Datums- und Uhrzeitangaben, die mir im Februar 2015 beim Schreiben dieses Kapitels angezeigt wurden. Sofern Sie keine Möglichkeit zur Zeitreise haben, werden bei Ihnen natürlich andere Ergebnisse angezeigt.

Mit Epochen-Zeitstempeln können Sie auch messen, wie lange es gedauert hat, um einen bestimmten Codeabschnitt auszuführen. Wenn Sie `time.time()` am Anfang und am Ende eines Codeblocks ausführen, können Sie anschließend den ersten Zeitstempel von dem zweiten subtrahieren und damit die Zeit ermitteln, die zwischen diesen Aufrufen vergangen ist. Um das auszuprobieren, geben Sie Folgendes in den Dateieditor ein:

```
import time  
def calcProd(): ❶  
    # Berechnet das Produkt der Zahlen von 1 bis 99.999  
    product = 1
```

```
for i in range(1, 100000):
    product = product * i
return product

startTime = time.time()      ❷
prod = calcProd()
endTime = time.time()        ❸
print('The result is %s digits long.' % (len(str(prod)))) ❹
print('Took %s seconds to calculate.' % (endTime - startTime)) ❺
```

Bei (❶) definierten wir die Funktion `calcProd()`, die alle ganzen Zahlen von 1 bis 99.999 durchläuft und ihr Produkt ausgibt. Dann rufen wir bei (❷) `time.time()` auf und speichern den Rückgabewert in `startTime`. Unmittelbar nach dem Aufruf von `calcProd()` rufen wir `time.time()` erneut auf, wobei wir das Ergebnis diesmal in `endTime` ablegen (❸). Am Ende geben wir die Länge des von `calcProd()` zurückgegebenen Produkts (❹) und die Ausführungszeit dieser Funktion aus (❺).

Speichern Sie das Programm als `calcProd.py` und führen Sie es aus. Dabei erhalten Sie folgende Ausgabe:

```
The result is 456569 digits long.
Took 2.844162940979004 seconds to calculate.
```

### Hinweis

Eine andere Möglichkeit, um die Ausführungszeit von Code zu messen, bietet die Funktion `cProfile.run()`, die weit ausführlichere Informationen bereitstellt als die einfache Technik mit `time.time()`. Eine Erklärung der Funktion `cProfile.run()` finden Sie auf <https://docs.python.org/3/library/profile.html>.

### Die Funktion `time.sleep()`

Wenn Sie ein Programm eine Weile anhalten müssen, rufen Sie die Funktion `time.sleep()` auf und übergeben Ihre gewünschte Wartezeit in Sekunden. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import time
>>> for i in range(3):
    print('Tick')    ❶
    time.sleep(1)   ❷
    print('Tock')   ❸
    time.sleep(1)   ❹
```

```

Tick
Tock
Tick
Tock
Tick
Tock
>>> time.sleep(5) ❸

```

Die for-Schleife gibt Tick aus (❶), hält eine Sekunde lang an (❷), gibt Tock aus (❸), hält eine weitere Sekunde lang an (❹), gibt wieder Tick aus, hält an usw., bis Tick und Tock jeweils dreimal ausgegeben wurden.

Die Funktion `time.sleep()` gibt die Steuerung nicht zurück, sondern *blockiert* die Ausführung von anderem Code, bis die übergebene Anzahl von Sekunden abgelaufen ist. Wenn Sie beispielsweise `time.sleep(5)` eingeben (❺), erscheint die nächste Eingabeaufforderung (>>>) erst, nachdem die fünf Sekunden verstrichen sind.

Beachten Sie, dass Sie die Ausführung von `time.sleep()` in IDLE nicht mit **Strg** + **C** abbrechen können. IDLE wartet erst die gesamte Pause ab, bevor die Ausnahme `KeyboardInterrupt` ausgelöst wird. Um dieses Problem zu umgehen, sollten Sie, um beispielsweise eine Pause von 30 Sekunden einzulegen, statt eines einzelnen Aufrufs von `time.sleep(30)` eine for-Schleife verwenden, um `time.sleep(1)` 30 Mal aufzurufen.

```
>>> for i in range(30):
    time.sleep(1)
```

Wenn Sie während dieser 30 Sekunden **Strg** + **C** drücken, wird die Ausnahme `KeyboardInterrupt` sofort ausgelöst.

## Zahlen runden

Bei der Arbeit mit Zeitangaben begegnen Ihnen oft Fließkommawerte mit sehr vielen Dezimalstellen. Um den Umgang mit diesen Werten zu vereinfachen, können Sie sie mithilfe der integrierten Python-Funktion `round()` verkürzen, die Fließkommazahlen auf die angegebene Genauigkeit rundet. Übergeben Sie einfach die zu rundende Zahl sowie ein optionales zweites Argument, das angibt, wie viele Stellen hinter dem Dezimalpunkt stehen bleiben sollen. Wenn Sie auf das zweite Argument verzichten, rundet `round()` die Zahl auf die nächste Ganzzahl. Das können Sie wie folgt in der interaktiven Shell ausprobieren:

```

>>> import time
>>> now = time.time()
>>> now
1425064108.017826

```

```
>>> round(now, 2)
1425064108.02
>>> round(now, 4)
1425064108.0178
>>> round(now)
1425064108
```

Hier importieren wir zunächst `time` und speichern den Wert von `time.time()` in `now`. Anschließend rufen wir `round(now, 2)` und `round(now, 4)` auf, um den Wert von `now` auf zwei bzw. vier Stellen hinter dem Dezimalpunkt zu runden. Mit `round(now)` schließlich runden wir den Wert auf die nächste Ganzzahl.

## Projekt: Superstoppuhr

Nehmen wir an, Sie wollen messen, wie viel Zeit Sie für langweilige Aufgaben aufwenden, die Sie noch nicht automatisiert haben. Leider haben Sie keine Stoppuhr und es ist auch erstaunlich schwierig, kostenlose Stoppuhr-Apps für Ihren Laptop oder Ihr Smartphone zu finden, zumindest keine, die nicht mit Werbung zugeplastert sind oder Ihren Browserverlauf an die Werbeindustrie petzen. (Dass das geschehen kann, steht in der Lizenzvereinbarung, die Sie akzeptiert haben. Sie haben die Lizenzvereinbarung doch gelesen, oder?) Allerdings können Sie in Python ein einfaches Stoppuhrprogramm schreiben.

Das Programm führt folgende Aufgaben aus:

- Die Zeit verfolgen, die zwischen zwei Betätigungen der Eingabetaste abläuft, wobei bei jedem dieser Tastendrücke ein neues Messintervall des Timers beginnt
- Die Nummer des Messintervalls, die Gesamtzeit und die im Intervall verstrichene Zeit ausgeben

Der Code muss also Folgendes tun:

- Die aktuelle Uhrzeit durch einen Aufruf von `time.time()` herausfinden und als Zeitstempel speichern. Dies geschieht sowohl beim Start des Programms als auch beim Start eines Messintervalls.
- Einen Zähler für die Messintervalle führen und den Wert jedes Mal um 1 erhöhen, wenn der Benutzer die Eingabetaste drückt
- Die verstrichene Zeit durch Subtraktion der Zeitstempel berechnen
- KeyboardInterrupt-Ausnahmen verarbeiten, damit der Benutzer das Programm über `Strg + C` abbrechen kann

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das Programm als `stopwatch.py`.

## Schritt 1: Das Programm auf die Zeitmessung vorbereiten

Da das Programm auf die aktuelle Uhrzeit zugreifen soll, müssen Sie das Modul `time` importieren. Vor dem Aufruf von `input()` soll es außerdem eine kurze Bedienungsanleitung ausgeben. Der Timer startet, nachdem der Benutzer die Eingabetaste gedrückt hat. Der Code beginnt dann, Zeitintervalle zu messen.

Geben Sie im Dateieditor den folgenden Code ein. Der `TODO`-Kommentar dient als Platzhalter für den noch fehlenden Code.

```
#! python3
# stopwatch.py - Ein einfaches Stoppuhrprogramm

import time

# Zeigt die Bedienungsanleitung für das Programm an
print('Press ENTER to begin. Afterwards, press ENTER to "click" the stop-
watch.')
print('Press Ctrl-C to quit.')
input() # Der Benutzer muss die Eingabetaste drücken, um zu beginnen
print('Started.')
startTime = time.time() # Ermittelt die Startzeit des ersten Intervalls
lastTime = startTime
lapNum = 1

# TODO: Zeitintervalle messen
```

Nach der Anzeige der Anweisungen wird das erste Zeitintervall ausgelöst. Dabei wird die Startzeit festgehalten und der Intervallzähler auf 1 gesetzt.

## Schritt 2: Intervalldauern messen und anzeigen

Als Nächstes schreiben wir Code, der ein neues Intervall startet, die Länge des vorherigen Intervalls und die Gesamtzeit seit dem Start des Stoppuhrprogramms berechnet, die Intervall- und die Gesamtzeit anzeigt und den Intervallzähler bei jedem neuen Intervall heraufsetzt. Ergänzen Sie das Programm um folgenden Code:

```
#! python3
# stopwatch.py - Ein einfaches Stoppuhrprogramm

import time

-- schnipp --
```

```
# Misst die Intervalldauern
try: ❶
    while True: ❷
        input()
        lapTime = round(time.time() - lastTime, 2) ❸
        totalTime = round(time.time() - startTime, 2) ❹
        print('Lap #{}: {} ({})'.format(lapNum, totalTime, lapTime), end='') ❺
        lapNum += 1
        lastTime = time.time() # Setzt die letzte Intervalldauer zurück
except KeyboardInterrupt: ❻
    # Verarbeitet Strg-C-Ausnahme, um die Fehlermeldung zu unterdrücken
    print('\nDone.')
```

Wenn der Benutzer **Strg** + **C** drückt, um das Stoppuhrprogramm abzubrechen, wird die Ausnahme `KeyboardInterrupt` ausgelöst. Um zu verhindern, dass diese Ausnahme zu einem Absturz des Programms führt, stellen wir diesen Teil des Programms in eine `try`-Anweisung (❶). Wird **Strg** + **C** gedrückt und die entsprechende Ausnahme ausgelöst, geht die Programmausführung zur `except`-Klausel über, wo wir festlegen, dass statt der Fehlermeldung zu `KeyboardInterrupt` lediglich `Done` ausgegeben werden soll (❻). Solange diese Tastenkombination nicht gedrückt wird, durchläuft das Programm eine Endlosschleife (❷), die `input()` aufruft und dann wartet, bis der Benutzer die Eingabetaste drückt, um das Intervall zu beenden. Wenn das geschieht, berechnen wir die Dauer des Intervalls, indem wir die `lastTime`, die Startzeit des Intervalls, von der aktuellen Zeit subtrahieren, die wir über `time.time()` ermitteln (❸). Durch Subtraktion der Startzeit des Stoppuhrprogramms (`startTime`) von der aktuellen Zeit ermitteln wir außerdem die Gesamlaufzeit (❹).

Da die Ergebnisse dieser Zeitberechnungen sehr viele Dezimalstellen aufweisen (z. B. 4.766272783279419), runden wir sie bei (❸) und (❹) mithilfe von `round()` auf zwei Stellen.

Bei (❺) geben wir die Intervallnummer, die Gesamlaufzeit und die Intervalldauer aus. Die Betätigung der Eingabetaste für den Aufruf von `input()` hat bereits einen Zeilenumbruch auf dem Bildschirm verursacht. Um einen doppelten Zeilenabstand in der Ausgabe zu vermeiden, übergeben wir der Funktion `print()` daher `end=''`. Nachdem wir die Informationen zu dem Intervall ausgegeben haben, bereiten wir das nächste Intervall vor. Dazu erhöhen wir den Intervallzähler `lapNum` um 1 und setzen `lastTime` auf die aktuelle Zeit, die somit zur Startzeit des neuen Intervalls wird.

## Vorschläge für ähnliche Programme

Eine Zeitmessung bietet viele Möglichkeiten für Ihre Programme. Für einige dieser Aufgaben können Sie auch fertige Apps herunterladen, doch wenn Sie die Pro-

gramme selbst schreiben, haben Sie den Vorteil, dass sie kostenlos und nicht mit Werbung und unnötigen Funktionen aufgeblättert sind. Beispielsweise können Sie Programme für folgende Aufgaben schreiben:

- Ein einfaches Stechuhrprogramm, das einen eingegebenen Personennamen erkennt und zum Ein- und Ausstempeln dieser Person jeweils die aktuelle Uhrzeit abruft
- Ein Programm um die Möglichkeit erweitern, die verstrichene Zeit seit dem Start eines Vorgangs zu ermitteln, z. B. seit dem Beginn eines Downloads mit dem Modul `requests` (siehe Kapitel 11)
- Regelmäßig prüfen, wie lange ein Programm schon läuft, und dem Benutzer die Möglichkeit bieten, Aufgaben abzubrechen, die zu viel Zeit in Anspruch nehmen

## Das Modul `datetime`

Das Modul `time` ist praktisch, wenn Sie mit dem Unix-Epochenzeitstempel arbeiten wollen, doch wenn Sie Kalenderdaten auf übersichtliche Weise anzeigen oder kalendarische Berechnungen durchführen wollen (z. B. um herauszufinden, welches Datum vor 205 Tagen war oder in 123 Tagen sein wird), bietet sich stattdessen das Modul `datetime` an.

Dieses Modul bringt seinen eigenen Datentyp mit, der ebenfalls `datetime` heißt. `datetime`-Werte stehen jeweils für eine bestimmte Zeitangabe. Das können Sie sich wie folgt in der interaktiven Shell ansehen:

```
>>> import datetime
>>> datetime.datetime.now()      ❶
datetime.datetime(2015, 2, 27, 11, 49, 55, 53)    ❷
>>> dt = datetime.datetime(2015, 10, 21, 16, 29, 0) ❸
>>> dt.year, dt.month, dt.day   ❹
(2015, 10, 21)
>>> dt.hour, dt.minute, dt.second ❺
(16, 29, 0)
```

Der Aufruf von `datetime.datetime.now()` (❶) gibt ein `datetime`-Objekt mit dem aktuellen Datum und der aktuellen Uhrzeit nach der Systemuhr des Computers zurück (❷). Dieses Objekt gibt das Jahr, den Monat, den Tag, die Stunde, die Minute, die Sekunde, die Millisekunde und die Mikrosekunde des aktuellen Augenblicks an. Mit der Funktion `datetime.datetime()` können Sie aber auch ein `datetime`-Objekt für einen bestimmten Zeitpunkt abrufen, indem Sie Integerwerte für Jahr, Monat, Tag, Stunde und Sekunde übergeben (❸). Diese Integerwerte werden in den Attributen `year`, `month`, `day` (❹), `hour`, `minute` und `second` (❺) des `datetime`-Objekts gespeichert.

Mit der Funktion `datetime.datetime.fromtimestamp()` können Sie einen Unix-Epochenzeitstempel in ein `datetime`-Objekt für die lokale Zeitzone umwandeln. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2015, 2, 27, 11, 13, 0, 604980)
```

Hier rufen wir die Funktion `datetime.datetime.fromtimestamp()` auf und übergeben ihr 10000000, woraufhin sie das `datetime`-Objekt für den Zeitpunkt 1.000.000 Sekunden nach der Unix-Epoche zurückgibt. Wenn Sie statt eines festen Werts `time.time()` übergeben, also den Unix-Epochenzeitstempel für den aktuellen Augenblick, erhalten Sie ein `datetime`-Objekt für den jetzigen Zeitpunkt. Die Ausdrücke `datetime.datetime.now()` und `datetime.datetime.fromtimestamp(time.time())` haben daher die gleiche Wirkung: Beide geben das `datetime`-Objekt für den augenblicklichen Zeitpunkt zurück.

### Hinweis

Die Beispiele wurden auf einem Computer ausgeführt, der auf Pacific Standard Time gesetzt ist. In einer anderen Zeitzone ergeben sich andere Resultate.

Um herauszufinden, ob ein Datum einem anderen vorausgeht, können Sie die betreffenden `datetime`-Objekte mithilfe von Vergleichsoperatoren vergleichen. Das spätere `datetime`-Objekt wird dabei als der »größere« Wert angesehen. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> halloween2015 = datetime.datetime(2015, 10, 31, 0, 0, 0) ❶
>>> newyears2016 = datetime.datetime(2016, 1, 1, 0, 0, 0) ❷
>>> oct31_2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
>>> halloween2015 == oct31_2015 ❸
True
>>> halloween2015 > newyears2016 ❹
False
>>> newyears2016 > halloween2015 ❺
True
>>> newyears2016 != oct31_2015
True
```

Hier erstellen wir zunächst ein `datetime`-Objekt für den ersten Augenblick (Mitternacht) des 31. Oktober 2015, das wir in `halloween2015` speichern (❶), und dann eines für den ersten Moment des 1. Januar 2016 in `newyear2016` (❷). Schließlich legen wir noch ein zweites `datetime`-Objekt für Mitternacht (0 Uhr) des 31. Oktober 2015 an und speichern es in `oct31_2015`. Der Vergleich von `halloween2015` und `oct31_2015`

zeigt, dass diese beiden Daten identisch sind (❸). Die nächsten Vergleiche ergeben, dass newyears2016 größer (also später) ist als halloween2015 (❹) (❺).

## Der Datentyp timedelta

Das Modul `datetime` verfügt auch über den Datentyp `timedelta`, der nicht für einen *Zeitpunkt* steht, sondern für eine *Zeitdauer*. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> delta = datetime.timedelta(days=11, hours=10, minutes=9, seconds=8) ❶
>>> delta.days, delta.seconds, delta.microseconds ❷
(11, 36548, 0)
>>> delta.total_seconds()
986948.0
>>> str(delta)
'11 days, 10:09:08'
```

Um ein `timedelta`-Objekt zu erstellen, verwenden Sie die Funktion `datetime.timedelta()`, die die Schlüsselwortargumente `weeks`, `days`, `hours`, `minutes`, `seconds`, `milliseconds` und `microseconds` entgegennimmt. Da Monate und Jahre unterschiedliche Dauern aufweisen können, gibt es keine Argumente für `month` und `year`. Ein `timedelta`-Objekt gibt die Gesamtdauer in Tagen, Sekunden und Mikrosekunden zurück, die in den Attributen `days`, `seconds` und `microseconds` gespeichert sind. Mit der Methode `total_seconds()` können Sie die Dauer in Sekunden ermitteln. Wenn Sie ein `timedelta`-Objekt an `str()` übergeben, erhalten Sie eine übersichtlich gestaltete Stringdarstellung des Objekts.

In diesem Beispiel übergeben wir `datetime.timedelta()` Schlüsselwortargumente, um eine Dauer von 11 Tagen, 10 Stunden, 9 Minuten und 8 Sekunden anzugeben, und speichern das zurückgegebene `timedelta`-Objekt in `delta` (❶). Im Attribut `days` dieses Objekts ist der Wert 11 gespeichert, im Attribut `seconds` der Wert 36548 (10 Stunden, 9 Minuten und 8 Sekunden ausgedrückt in Sekunden) (❷). Der Aufruf von `total_seconds()` ergibt, dass 11 Tage, 10 Stunden, 9 Minuten und 8 Sekunden einer Dauer von 986.948 Sekunden entsprechen. Um einen String mit einer übersichtlichen Darstellung der Dauer zu erhalten, übergeben wir das `timedelta`-Objekt schließlich an `str()`.

Für die *Kalenderarithmetik* mit `datetime`-Werten können Sie die arithmetischen Operatoren verwenden. Um beispielsweise zu berechnen, welches Datum wir in 1000 Tagen haben werden, geben Sie Folgendes in die interaktive Shell ein:

```
>>> dt = datetime.datetime.now()
>>> dt
datetime.datetime(2015, 2, 27, 18, 38, 50, 636181)
>>> thousandDays = datetime.timedelta(days=1000)
>>> dt + thousandDays
datetime.datetime(2017, 11, 23, 18, 38, 50, 636181)
```

Als Erstes erstellen Sie hier ein `datetime`-Objekt für den aktuellen Zeitpunkt und speichern es in `dt`. Dann erzeugen Sie ein `timedelta`-Objekt für die Dauer von 1000 Tagen, das Sie in `thousandDays` speichern. Wenn Sie `dt` und `thousandDays` addieren, erhalten Sie ein `datetime`-Objekt für das Datum in 1000 Tagen. Python kümmert sich um die Berechnungen und findet heraus, dass wir 1000 Tage nach dem 27. Februar 2015 den 23. November 2017 haben werden. Das ist sehr praktisch, denn wenn Sie selbst von einem gegebenen Datum aus 1000 Tage in die Zukunft rechnen wollten, müssten Sie die verschiedenen Monatslängen, Schaltjahre und andere knifflige Einzelheiten berücksichtigen. Das Modul `datetime` nimmt Ihnen diese Arbeit ab.

Sie können auch `timedelta`-Objekte zu `datetime`-Objekten oder anderen `timedelta`-Objekten addieren oder von ihnen subtrahieren. Dazu verwenden Sie die Operatoren `+` und `-`. Mit den Operatoren `*` und `/` ist es überdies möglich, `timedelta`-Objekte mit Integer- oder Fließkommawerten zu multiplizieren oder durch sie zu dividieren:

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)     ❶
>>> aboutThirtyYears = datetime.timedelta(days=365 * 30)   ❷
>>> oct21st
datetime.datetime(2015, 10, 21, 16, 29)
>>> oct21st - aboutThirtyYears
datetime.datetime(1985, 10, 28, 16, 29)
>>> oct21st - (2 * aboutThirtyYears)
datetime.datetime(1955, 11, 5, 16, 29)
```

Hier erstellen wir ein `datetime`-Objekt für den 21. Oktober 2015 (❶) und ein `timedelta`-Objekt für die Dauer von ca. 30 Jahren (wobei wir für alle Jahre eine Länge von 365 Tagen ansetzen) (❷). Wenn wir `aboutThirtyYears` von `oct21st` abziehen, erhalten wir ein `datetime`-Objekt für das Datum 30 Jahre vor dem 21. Oktober 2015. Die Subtraktion von `2 * aboutThirtyYears` von `oct21st` ergibt ein `datetime`-Objekt für den Zeitpunkt 60 Jahre vor dem Stichtag.

## Anhalten bis zu einem bestimmten Zeitpunkt

Mit der Methode `time.sleep()` können Sie ein Programm für eine gewisse Dauer anhalten. Wenn Sie eine `while`-Schleife verwenden, können Sie auch eine Pause bis zu einem bestimmten Zeitpunkt verlangen. Beispielsweise wird die Schleife in dem folgenden Code bis Halloween 2016 durchlaufen:

```
import datetime
import time
halloween2016 = datetime.datetime(2016, 10, 31, 0, 0, 0)
while datetime.datetime.now() < halloween2016:
    time.sleep(1)
```

Der Aufruf von `time.sleep(1)` hält das Python-Programm an, damit der Computer keine CPU-Verarbeitungszyklen darauf verschwendet, ständig die Zeit zu ermitteln. Stattdessen prüft die `while`-Schleife die Bedingung einmal pro Sekunde und fährt mit dem Rest des Programms nach Halloween 2016 fort.

### **datetime-Objekte in Strings umwandeln**

Epochenzzeitstempel und `datetime`-Objekte sind für Menschen nicht gerade übersichtlich. Mit der Methode `strftime()` können Sie jedoch ein `datetime`-Objekt als String darstellen. (Das `f` in dem Methodennamen steht übrigens für *format*.)

Diese Methode verwendet ähnliche Direktiven, wie sie auch bei der Stringformatierung in Python zum Tragen kommen. Eine Liste dieser Direktiven finden Sie in Tabelle 15–1.

Direktive	Bedeutung
<code>%Y</code>	Jahr mit Jahrhundertangabe, z. B. '2014'
<code>%y</code>	Jahr ohne Jahrhundertangabe, also '00' bis '99' (1970 bis 2069)
<code>%m</code>	Monat als Dezimalzahl, '01' bis '12'
<code>%B</code>	Ausgeschriebener Monatsname, z. B. 'November'
<code>%b</code>	Abgekürzter Monatsname, z. B. 'Nov'
<code>%d</code>	Tag im Monat, '01' bis '31'
<code>%j</code>	Tag im Jahr, '001' bis 366'
<code>%w</code>	Tag in der Woche, '0' (Sonntag) bis '6' (Samstag)
<code>%A</code>	Ausgeschriebener Name des Wochentags, z. B. 'Monday'
<code>%a</code>	Abgekürzter Name des Wochentags, z. B. 'Mon'
<code>%H</code>	Stunde (24-Stunden-Zählung), '00' bis '23'
<code>%I</code>	Stunde (12-Stunden-Zählung), '00' bis '12'
<code>%M</code>	Minute, '00' bis '59'
<code>%S</code>	Sekunde, '00' bis '59'
<code>%p</code>	Angabe für Vormittag oder Nachmittag, 'AM' oder 'PM'
<code>%%</code>	Maskierungssequenz für das Zeichen '%'

**Tab. 15–1** Direktiven für `strftime()`

Wenn Sie `strftime()` einen String mit Formatierungsdirektiven (einschließlich jeglicher gewünschter Schrägstriche, Doppelpunkte usw.) übergeben, so gibt diese Funktion die Informationen aus dem `datetime`-Objekt als formatierten String zurück:

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d %H:%M:%S')
'2015/10/21 16:29:00'
>>> oct21st.strftime('%I:%M %p')
'04:29 PM'
>>> oct21st.strftime("%B of '%y")
"October of '15"
```

Hier haben wir das `datetime`-Objekt für 16.29 Uhr am 21. Oktober 2015 in `oct21st` gespeichert. Wenn wir dafür `strftime()` aufrufen und den Formatierungsstring '`%Y/%m/%d %H:%M:%S`' übergeben, erhalten wir einen String mit den Werten 2015, 10 und 21 getrennt durch Schrägstriche und 16, 29 und 00 getrennt durch Doppelpunkte zurück. Beim Formatierungsstring '`%I:%M %p`' erhalten wir '`04:29 PM`' zurück, bei "`%B of '%y`" dagegen "`October of '15`". Beachten Sie, dass Sie `strftime()` nicht `datetime.datetime` voranstellen.

### Strings in `datetime`-Objekte umwandeln

Wenn Sie einen String mit Datumsinformationen wie '`2015/10/21 16:29:00`' oder '`October 21, 2015`' in ein `datetime`-Objekt umwandeln müssen, verwenden Sie die Funktion `datetime.datetime.strptime()`, die Umkehrfunktion zu `strftime()` (wobei das `p` für *parse* steht). Damit `strptime()` weiß, wie der String aufgebaut ist, müssen Sie ihr einen Formatierungsstring übergeben, der aus den gleichen Direktiven aufgebaut ist wie der Formatierungsstring für `strftime()`.

Das können Sie sich wie folgt in der interaktiven Shell ansehen:

```
>>> datetime.datetime.strptime('October 21, 2015', '%B %d, %Y')      ❶
datetime.datetime(2015, 10, 21, 0, 0)
>>> datetime.datetime.strptime('2015/10/21 16:29:00', '%Y/%m/%d %H:%M:%S')
datetime.datetime(2015, 10, 21, 16, 29)
>>> datetime.datetime.strptime("October of '15", "%B of '%y")
datetime.datetime(2015, 10, 1, 0, 0)
>>> datetime.datetime.strptime("November of '63", "%B of '%y")
datetime.datetime(2063, 11, 1, 0, 0)
```

Um aus dem String '`October 21, 2015`' ein `datetime`-Objekt zu gewinnen, müssen Sie `strptime()` als erstes Argument den Datumsstring übergeben und als zweites einen Formatierungsstring, der den Aufbau dieser Datumsangabe beschreibt (❶). Wenn die Beschreibung nicht genau zu dem Datumsstring passt, löst Python die Ausnahme `ValueError` aus.

## Die Zeitfunktionen von Python im Überblick

Bei der Arbeit mit Datums- und Uhrzeitangaben kommen in Python eine ganze Reihe verschiedener Datentypen und Funktionen ins Spiel. Die folgende Übersicht zeigt, wie Sie die drei verschiedenen Arten von Werten zur Darstellung von Zeitangaben einsetzen:

- Der Unix-Epochenzeitstempel (der vom Modul `time` verwendet wird) ist ein Fließkomma- oder Integerwert, der die Anzahl der seit 0 Uhr des 1. Januar 1970 UTC verstrichenen Sekunden angibt.
- In einem `datetime`-Objekt (verwendet vom Modul `datetime`) sind Integerwerte in den Attributen `year`, `month`, `day`, `hour`, `minute` und `second` gespeichert.
- Ein `timedelta`-Objekt (verwendet vom Modul `datetime`) stellt keinen bestimmten Zeitpunkt dar, sondern eine Zeitdauer.

Die folgende Übersicht nennt die verschiedenen Funktionen für Zeitangaben mit ihren Parametern und Rückgabewerten:

- Die Funktion `time.time()` gibt einen Fließkommawert des Epochenzeitstempels für den aktuellen Zeitpunkt zurück.
- Die Funktion `time.sleep(seconds)` hält das Programm für die im Argument `seconds` angegebene Dauer in Sekunden an.
- Die Funktion `datetime.datetime(year, month, day, hour, minute, second)` gibt das `datetime`-Objekt für den in den Argumenten angegebenen Zeitpunkt zurück. Wenn die Argumente `hour`, `minute` oder `second` nicht angegeben sind, wird der Standardwert 0 angenommen.
- Die Funktion `datetime.datetime.now()` gibt das `datetime`-Objekt für den aktuellen Zeitpunkt zurück.
- Die Funktion `datetime.datetime.fromtimestamp(epoch)` gibt das `datetime`-Objekt für den Zeitpunkt zurück, der durch das Argument `epoch` in Form eines Zeitstempels angegeben ist.
- Die Funktion `datetime.timedelta(weeks, days, hours, minutes, seconds, milliseconds, microseconds)` gibt ein `timedelta`-Objekt für die angegebene Zeitdauer zurück. Alle Schlüsselwortargumente dieser Funktion sind optional. Es gibt keine Argumente für Monate und Jahre.
- Die Methode `total_seconds()` für `timedelta`-Objekte gibt die von dem `timedelta`-Objekt dargestellte Dauer als Angabe in Sekunden zurück.
- Die Methode `strftime(format)` liefert einen String, der die Zeitangabe des `datetime`-Objekts in der mit dem String `format` angegebenen Gestaltung darstellt. Einzelheiten zur Formatierung finden Sie in Tabelle 15–1.
- Die Funktion `datetime.datetime.strptime(time_string, format)` gibt ein `datetime`-Objekt für den in `time_string` angegebenen Zeitpunkt zurück. Das Argument `format` gibt dabei an, wie das Datum in `time_string` dargestellt wird. Einzelheiten zur Formatierung finden Sie in Tabelle 15–1.

## Multithreading

Zur Einführung in Multithreading sehen wir uns ein Beispiel an. Nehmen Sie an, Sie möchten dafür sorgen, dass bestimmter Code nach einer Verzögerung oder an einem bestimmten Zeitpunkt ausgeführt wird. Dazu können Sie am Anfang des Programms Folgendes schreiben:

```
import time, datetime

startTime = datetime.datetime(2029, 10, 31, 0, 0, 0)
while datetime.datetime.now() < startTime:
    time.sleep(1)

print('Program now starting on Halloween 2029')
-- schnipp --
```

Dieser Code legt als Startzeit den 31. Oktober 2029 fest und ruft danach immer wieder `time.sleep(1)` auf, bis dieser Zeitpunkt gekommen ist. Während das Programm darauf wartet, dass die Schleife mit dem Aufruf von `time.sleep()` beendet ist, kann das Programm nichts anderes tun, sondern dreht bis Halloween 2029 einfach Däumchen. Das liegt daran, dass Python-Programme standardmäßig nur über einen einzigen *Ausführungsthread* verfügen.

Was ist ein Ausführungsthread? Bei der Erörterung der Flusssteuerung in Kapitel 2 habe ich den Programmablauf damit verglichen, dass Sie mit dem Finger den Codezeilen in einem Programm folgen und jeweils zur nächsten Zeile oder dorthin vorrücken, wohin eine Flusssteuerungsanweisung verweist. Ein *Single-Thread-* oder *Einzelthread*-Programm hat nur einen Finger, ein *Multithread-Programm* dagegen mehrere. Jeder dieser Finger rückt dabei immer zur nächsten Codezeile vor, die die Steueranweisungen angeben, allerdings können sich die einzelnen Finger an verschiedenen Stellen im Programm befinden und unterschiedliche Codezeilen gleichzeitig ausführen. (Alle bisher in diesem Buch beschriebenen Programme wiesen nur einen Thread auf.)

Anstatt das gesamte Programm darauf warten zu lassen, dass die Funktion `time.sleep()` zum Ende kommt, können Sie den Code, der um eine bestimmte Zeit oder bis zu einem bestimmten Datum aufgeschoben werden soll, mithilfe des Python-Moduls `threading` in einen eigenen Thread stellen. Dieser Thread wird dann angehalten, solange `time.sleep` aufgerufen wird. In der Zwischenzeit kann das Programm im ursprünglichen Thread andere Aufgaben erfüllen.

Um einen neuen Thread zu erstellen, müssen Sie ein Thread-Objekt anlegen, indem Sie die Funktion `threading.Thread()` aufrufen. Geben Sie folgenden Code im Dateieditor ein und speichern Sie ihn als `threadDemo.py`:

```
import threading, time
print('Start of program.')

def takeANap():    ❶
    time.sleep(5)
    print('Wake up!')

threadObj = threading.Thread(target=takeANap) ❷
threadObj.start() ❸

print('End of program.)
```

Bei (❶) definieren wir eine Funktion, die wir in einem neuen Thread platzieren wollen. Um das Thread-Objekt zu erstellen, rufen wir `threading.Thread()` auf und übergeben das Schlüsselwortargument `target=takeANap` (❷). Damit geben wir an, dass wir in diesem neuen Thread die Funktion `takeANap()` aufrufen möchten. Beachten Sie aber, dass Sie als Argument `target=takeANap` schreiben müssen, nicht `target=takeANap()`, weil Sie die Funktion selbst als Argument übergeben. Bei der Schreibweise mit den Klammern würden Sie die Funktion aufrufen und ihren Rückgabewert übergeben.

Nachdem wir das neue Thread-Objekt in `threadObj` gespeichert haben, rufen wir `threadObj.start()` auf (❸), um den neuen Thread zu erstellen und darin die angegebene Funktion auszuführen. Die Ausgabe des Programms sieht wie folgt aus:

```
Start of program.
End of program.
Wake up!
```

Das wirkt ein bisschen eigenartig. Da `print('End of program.')` die letzte Zeile des Programms ist, sollte man meinen, dass diese Meldung auch als Letztes ausgegeben wird. Dass `Wake up!` erst danach erscheint, liegt daran, dass beim Aufruf von `threadObj.start()` die angegebene Funktion in einem neuen Thread ausgeführt wird. Während der Hauptthread mit `print('End of program.')` fortfährt, führt der neue Thread `time.sleep(5)` aus und wartet daher fünf Sekunden lang. Erst nach dem Aufwachen aus diesem Nickerchen gibt er '`Wake up!`' aus und beendet die Funktion `takeANap()`. Daher gibt das Programm '`Wake up!`' als Letztes aus.

Normalerweise endet ein Programm, wenn die letzte Codezeile in der Datei ausgeführt wurde (oder die Funktion `sys.exit()` aufgerufen wurde), doch `thread-Demo.py` weist zwei Threads auf. Der erste ist der ursprüngliche Thread, der mit dem Programmstart beginnt und nach `print('End of program.')` endet. Der zweite wird beim Aufruf von `threadObj.start()` erstellt, beginnt mit dem Start der Funktion `takeANap()` und endet, wenn diese die Steuerung zurückgibt.

Ein Python-Programm endet erst, wenn alle Threads beendet sind. Bei *threadDemo.py* war nach dem Ende des ersten Threads der zweite noch aktiv, da er den Aufruf von `time.sleep(5)` ausführte.

## Argumente an die Zielfunktion eines Threads übergeben

Wenn die Zielfunktion, die Sie in einem neuen Thread ausführen wollen, Argumente entgegennimmt, können Sie diese an `threading.Thread()` übergeben. Nehmen wir an, Sie wollen folgenden Aufruf von `print()` in einem eigenen Thread ausführen:

```
>>> print('Cats', 'Dogs', 'Frogs', sep=' & ')
    Cats & Dogs & Frogs
```

Dieser Aufruf nimmt die drei regulären Argumente `'Cats'`, `'Dogs'` und `'Frogs'` sowie das Schlüsselwortargument `sep=' & '` entgegen. Die regulären Argumente können Sie als Liste im Schlüsselwortargument `args` von `threading.Thread()` übergeben. Für das Schlüsselwortargument verwenden Sie ein Dictionary im Schlüsselwortargument `kwargs` von `threading.Thread()`.

Das können Sie wie folgt in der interaktiven Shell veranschaulichen:

```
>>> import threading
>>> threadObj = threading.Thread(target=print, args=['Cats', 'Dogs',
   'Frogs'], kwargs={'sep': ' & '})
>>> threadObj.start()
    Cats & Dogs & Frogs
```

Um die Argumente `'Cats'`, `'Dogs'` und `'Frogs'` und das Schlüsselwortargument `sep=' & '` an `print()` im neuen Thread zu übergeben, übergeben wir `args=['Cats', 'Dogs', 'Frogs']` und `kwargs={'sep': ' & '}` an `threading.Thread()`.

Der Aufruf von `threadObj.start()` erstellt einen neuen Thread, in dem die Funktion `print()` aufgerufen wird, und übergibt dabei die Argumente `'Cats'`, `'Dogs'` und `'Frogs'` sowie das Schlüsselwortargument `sep` mit dem Wert `' & '`.

Die folgende Schreibweise für den Aufruf von `print()` in einem neuen Thread ist dagegen falsch:

```
threadObj = threading.Thread(target=print('Cats', 'Dogs', 'Frogs', sep=' & '))
```

Dadurch übergeben Sie jedoch nicht die Funktion `print()`, sondern Sie rufen sie auf und übergeben ihren Rückgabewert (der stets `None` ist) als Schlüsselwortargument `target`. Um Argumente an eine Funktion in einem neuen Thread zu übergeben, müssen Sie die Schlüsselwortargumente `args` und `kwargs` der Funktion `threading.Thread()` verwenden.

## Probleme der Nebenläufigkeit

Es ist ganz einfach, mehrere neue Threads zu erstellen und alle gleichzeitig ausführen zu lassen. Allerdings können bei einer solchen *Nebenläufigkeit* von mehreren Threads Probleme auftreten, etwa wenn verschiedene Threads gleichzeitig in ein und derselben Variablen lesen und schreiben. Solche Nebenläufigkeitsprobleme lassen sich nur schwer reproduzieren und sind daher nicht so einfach zu beheben.

Die Multithread-Programmierung ist ein vielschichtiges Thema, das in diesem Buch nicht erschöpfend behandelt werden kann. Merken Sie sich aber Folgendes: Um Nebenläufigkeitsprobleme zu vermeiden, müssen Sie verhindern, dass mehrere Threads gleichzeitig lesend oder schreibend auf dieselbe Variable zugreifen. Wenn Sie ein neues Thread-Objekt erstellen, müssen Sie dafür sorgen, dass seine Zielfunktion nur lokale Variablen verwendet. Dadurch können Sie schwer zu findende Nebenläufigkeitsprobleme in Ihren Programmen verhindern.

### Hinweis

Einen Einsteigerkurs in Multithread-Programmierung finden Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).

## Projekt: Multithread-Version des XKCD-Download-Programms

In Kapitel 11 haben Sie ein Programm geschrieben, das alle Comics von der Website XKCD herunterlädt. Dieses Programm verfügte nur über einen einzigen Thread und konnte daher immer nur einen Comic auf einmal herunterladen. Ein Großteil der Programmlaufzeit wurde dafür verwendet, die Netzwerkverbindung zu Beginn des Downloads aufzubauen und die heruntergeladenen Bilder auf die Festplatte zu schreiben. Wenn Sie über eine Breitband-Internetverbindung verfügen, nutzt dieses Programm die vorhandene Bandbreite nicht komplett aus.

Eine bessere Ausnutzung der Internetverbindung und einen schnelleren Download der gesamten Comics können Sie mit einem Multithread-Programm erzielen, bei dem einige Threads die Comics herunterladen, während andere schon die nächsten Verbindungen herstellen und die bereits heruntergeladenen Bilder auf die Festplatte schreiben. Öffnen Sie ein Dateieditorfenster und speichern Sie das neue Programm als *multidownloadXkcd.py*. Darin wandeln Sie das ursprüngliche Programm so ab, dass es mehrere Threads nutzt. Den kompletten Quellcode der neuen Version finden Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).

## Schritt 1: Eine Funktion für den Download verwenden

Der Download-Code in diesem Programm ist größtenteils identisch mit dem aus Kapitel 11, weshalb ich hier nicht noch einmal auf requests und Beautiful Soup eingehe. Die wichtigsten Änderungen bestehen darin, dass Sie das Modul threading importieren und die Funktion `downloadXkcd()` definieren müssen. Die Parameter dieser Funktion sind die Anfangs- und Endnummer des Bereichs für die herunterzuladenden Comics.

Beispielsweise durchläuft `downloadXkcd(140, 280)` den Download-Code, um die Comics `http://xkcd.com/140` bis `http://xkcd.com/279` abzurufen. Jeder Thread, den Sie erstellen, ruft `downloadXkcd()` auf und übergibt ihr einen anderen Satz von Comics.

Geben Sie folgenden Code für das Programm `multidownloadXkcd.py` ein:

```
#! python3
# multidownloadXkcd.py - Lädt XKCD-Comics in mehreren Threads herunter

import requests, os, bs4, threading
os.makedirs('xkcd', exist_ok=True) # Speichert Comics in ./xkcd ❶

def downloadXkcd(startComic, endComic): ❷
    for urlNumber in range(startComic, endComic): ❸
        # Lädt die Seite herunter
        print('Downloading page http://xkcd.com/%s...' % (urlNumber))
        res = requests.get('http://xkcd.com/%s' % (urlNumber)) ❹
        res.raise_for_status()

        soup = bs4.BeautifulSoup(res.text) ❺

        # Findet den URL für den Satz von Comics
        comicElem = soup.select('#comic img') ❻
        if comicElem == []:
            print('Could not find comic image.')
        else:
            comicUrl = comicElem[0].get('src') ❼
            # Lädt das Bild herunter
            print('Downloading image %s...' % (comicUrl))
            res = requests.get(comicUrl) ❽
            res.raise_for_status()

            # Speichert das Bild in ./xkcd.
            imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)),
                            'wb')
            for chunk in res.iter_content(100000):
                imageFile.write(chunk)
            imageFile.close()

    # TODO: Thread-Objekte erstellen und starten
    # TODO: Auf das Ende aller Threads warten
```

Nach dem Import der erforderlichen Module erstellen wir das Verzeichnis, in dem die Comics gespeichert werden sollen (❶), und definieren `downloadXkcd()` (❷). Darin durchlaufen wir alle Zahlen aus dem angegebenen Bereich (❸) und laden jede Seite herunter (❹). Mithilfe von Beautiful Soup durchsuchen wir den HTML-Text jeder Seite (❺), um das Bild des Comics zu finden (❻). Ist auf der Seite kein Bild zu finden, geben wir eine Meldung aus. Andernfalls rufen wir den URL des Bilds ab (❾) und laden es herunter (❿). Am Ende speichern wir das Bild in dem vorgesehenen Verzeichnis.

## Schritt 2: Threads erstellen und starten

Nun können wir mehrere Threads erstellen, die alle `downloadXkcd()` aufrufen, um verschiedene Gruppen von Comics von der XKCD-Website herunterzuladen. Fügen Sie in `multidownloadXkcd.py` hinter der Definition von `downloadXkcd()` folgenden Code hinzu:

```
#! python3
# multidownloadXkcd.py - Lädt XKCD-Comics in mehreren Threads herunter

-- schnipp --

# Erstellt und startet Thread-Objekte
downloadThreads = []          # Liste aller Thread-Objekte
for i in range(0, 1400, 100):  # Läuft 14 Mal und erstellt 14 Threads
    downloadThread = threading.Thread(target=downloadXkcd, args=(i, i + 99))
    downloadThreads.append(downloadThread)
    downloadThread.start()
```

Als Erstes erstellen wir die leere Liste `downloadThreads`, mit der wir uns später über die erstellten Thread-Objekte auf dem neuesten Stand halten. Danach starten wir die `for`-Schleife. Bei jedem Durchlauf erstellen wir mit `threading.Thread()` ein Thread-Objekt, nehmen es in die Liste auf und rufen `start()` auf, um `downloadXkcd()` in dem neuen Thread auszuführen. Die Variable `i` der `for`-Schleife durchläuft den Bereich von 0 bis 1400 in Schritten von 100, sodass `i` beim ersten Durchlauf 0 beträgt, beim zweiten 100, beim dritten 200 usw. Da wir `args=(i, i + 99)` an `threading.Thread()` übergeben, erhält `downloadXkcd()` beim ersten Durchlauf die Argumente 0 und 99, beim zweiten 100 und 199 usw.

Während die Methode `start()` des Thread-Objekts aufgerufen wird und der neue Thread den Code in `downloadXkcd()` auszuführen beginnt, läuft der Hauptthread weiter und durchläuft abermals die `for`-Schleife, um den nächsten Thread zu erstellen.

### Schritt 3: Auf das Ende aller Threads warten

Der Hauptthread fährt ganz normal fort, während die anderen Threads Comics herunterladen. Nehmen wir aber an, dass Sie im Hauptthread noch Code haben, der erst ausgeführt werden soll, wenn alle anderen Threads abgeschlossen sind. Wenn Sie die Methode `join()` eines Thread-Objekts aufrufen, wird der aufrufende Thread blockiert, bis der andere beendet ist. Wenn Sie in einer `for`-Schleife alle Thread-Objekte in der Liste `downloadThreads` durchlaufen, kann der Hauptthread `join()` für alle anderen Threads aufrufen. Fügen Sie am Ende Ihres Programms folgenden Code hinzu:

```
#! python3
# multidownloadXkcd.py - Lädt XKCD-Comics in mehreren Threads herunter

-- schnipp --

# Wartet, bis alle Threads beendet sind
for downloadThread in downloadThreads:
    downloadThread.join()
print('Done.')
```

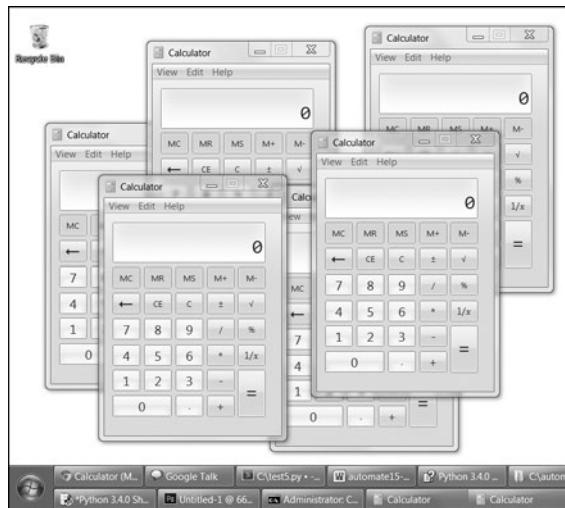
Der String `'Done.'` wird erst ausgegeben, wenn alle `join()`-Aufrufe die Steuerung zurückgegeben haben. Wenn ein Thread beim Aufruf seiner `join()`-Methode bereits beendet ist, dann gibt die Methode die Steuerung sofort zurück. Falls Sie Ihr Programm mit Code erweitern wollen, der erst nach dem Herunterladen aller Comics ausgeführt werden soll, verwenden Sie diesen Code anstelle der Zeile `print('Done.').`.

## Andere Programme von Python aus starten

Mit der Funktion `Popen()` aus dem integrierten Modul `subprocess` kann ein Python-Programm andere Programme auf Ihrem Computer starten. (Das `P` im Namen der Funktion steht dabei für *process*.) Wenn Sie mehrere Instanzen einer Anwendung geöffnet haben – z. B. mehrere Fenster, in denen Ihr Webbrowser läuft –, so ist jede davon ein eigenständiger Prozess desselben Programms. In dem Beispiel aus Abb. 15–1 sind mehrere Prozesse des Programms *Rechner* zur gleichen Zeit geöffnet.

Jeder Prozess wiederum kann aus mehreren Threads bestehen. Anders als Threads können Prozesse die Variablen anderer Prozesse nicht direkt lesen oder schreiben. Um in unserem Bild zu bleiben: Wenn bei einem Multithread-Programm mehrere Finger dem Quellcode folgen, dann können Sie sich mehrere Prozesse desselben Programms so vorstellen, als ob ein Freund mit einem eigenen Exemplar des Quellcodes neben Ihnen sitzt. Dabei führen Sie beide unabhängig voneinander das gleiche Programm aus.

Um in einem Python-Skript ein externes Programm zu starten, müssen Sie dessen Dateinamen an `subprocess.Popen()` übergeben. (Um den Dateinamen einer Anwendung in Erfahrung zu bringen, rechtsklicken Sie unter Windows auf ihren Startmenüeintrag und wählen *Eigenschaften*. Unter OS X klicken Sie bei gedrückter **[ctrl]**-Taste auf die Anwendung und wählen *Paketinhalt anzeigen*. In dem daraufhin geöffneten Dialogfeld können Sie den Pfad zu der ausführbaren Datei ablesen.) Die Funktion `Popen()` gibt die Steuerung anschließend sofort zurück. Das gestartete Programm wird in einem anderen Thread ausgeführt als das Python-Programm.



**Abb. 15–1** Sechs laufende Prozesse des Programms Rechner

Geben Sie auf einem Windows-Computer Folgendes in die interaktive Shell ein:

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
<subprocess.Popen object at 0x000000003055A58>
```

Auf Ubuntu Linux können Sie das wie folgt ausprobieren:

```
>>> import subprocess
>>> subprocess.Popen('/usr/bin/gnome-calculator')
<subprocess.Popen object at 0x7f2bcf93b20>
```

Auf einem OS X-Computer müssen Sie eine andere Vorgehensweise anwenden. Mehr dazu erfahren Sie im Abschnitt »Dateien in ihren Standardanwendungen öffnen« weiter hinten in diesem Kapitel.

Der Rückgabewert der Funktion ist ein Popen-Objekt mit den beiden nützlichen Methoden `poll()` und `wait()`.

Wenn Sie `poll()` verwenden, ist das ungefähr so, als würden Sie Ihren Freund fragen, ob er damit fertig ist, seinem Exemplar des Codes mit dem Finger zu folgen. Diese Methode gibt `None` zurück, wenn der Prozess beim Aufruf von `poll()` immer noch läuft. Ist das betreffende Programm dagegen beendet, wird ein Integerwert mit dem *Beendigungscode* des Prozesses zurückgegeben. Dieser Code gibt an, ob der Prozess ohne Fehler beendet wurde (Code 0) oder ob ein Fehler aufgetreten ist (gewöhnlich 1, allerdings kann je nach Programm auch ein anderer von null verschiedener Wert verwendet werden).

Die Wirkung von `wait()` können Sie sich so vorstellen, als ob Sie darauf warten, bis Ihr Freund mit der Arbeit an seinem Exemplar des Codes fertig ist, bevor Sie selbst mit Ihrem Exemplar weiterarbeiten. Diese Methode blockiert das laufende Programm, bis der gestartete Prozess beendet ist. Das ist nützlich, wenn Ihr Programm anhalten soll, bis der Benutzer seine Arbeit in dem anderen Programm beendet hat. Der Rückgabewert von `wait()` ist abermals der Beendigungscode des Prozesses.

Unter Windows können Sie das wie folgt in der interaktiven Shell ausprobieren. Der Aufruf von `wait()` blockiert die weitere Ausführung, bis Sie das gestartete Taschenrechnerprogramm beenden.

```
>>> calcProc = subprocess.Popen('c:\\Windows\\System32\\calc.exe')    ❶
>>> calcProc.poll() == None    ❷
True
>>> calcProc.wait()    ❸
0
>>> calcProc.poll()
0
```

Hier öffnen wir als Erstes einen Prozess des Taschenrechnerprogramms (❶). Während dieses Programm läuft, prüfen wir, ob `poll()` den Wert `None` zurückgibt (❷). Das sollte auch tatsächlich der Fall sein, da der Prozess schließlich immer noch läuft. Danach schließen wir das Taschenrechnerprogramm und rufen für den beendeten Prozess `wait()` auf (❸). Jetzt geben sowohl `wait()` als auch `poll()` den Wert 0 zurück, was bedeutet, dass der Prozess ohne Fehler beendet wurde.

## Befehlszeilenargumente an `Popen()` übergeben

Für die Prozesse, die Sie mit `Popen()` öffnen, können Sie auch Befehlszeilenargumente bereitstellen, und zwar in Form einer Liste, die Sie als ein einziges Argument an `Popen()` übergeben. Der erste String in dieser Liste ist wie gehabt der Name der ausführbaren Datei des zu startenden Programms; alle anderen sind die Argu-

mente, die Sie dem gewünschten Programm übergeben möchten. Diese Liste ist also mit dem Wert von `sys.argv` für das gestartete Programm identisch.

Befehlszeilenargumente werden bei grafischen Programmen weniger intensiv genutzt als bei Befehlszeilenprogrammen. Allerdings nehmen die meisten von ihnen einen Dateinamen als Argument entgegen, um diese Datei unmittelbar in der Anwendung zu öffnen. Wenn Sie beispielsweise unter Windows eine Textdatei namens `C:\hello.txt` haben, können Sie in der interaktiven Shell Folgendes eingeben:

```
>>> subprocess.Popen(['C:\\Windows\\\\notepad.exe', 'C:\\hello.txt'])
<subprocess.Popen object at 0x0000000032DCEB8>
```

Dadurch wird nicht nur der Windows-Editor (*Notepad*) gestartet, sondern darin auch sofort `C:\hello.txt` geöffnet.

### Taskplaner, launchd und cron

Wenn Sie im Umgang mit Computern versiert sind, kennen Sie vielleicht schon den Taskplaner in Windows, `launchd` in OS X oder `cron` in Linux. Mit diesen gut dokumentierten und zuverlässigen Einrichtungen können Sie Zeitpläne aufstellen, um Anwendungen zu vorher festgelegten Zeitpunkten zu starten. Auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) finden Sie Links zu Tutorials, in denen Sie mehr darüber lernen können.

Die eingebauten Zeitplanungswerkzeuge des Betriebssystems zu verwenden, erspart es Ihnen, eigenen Code zur Überprüfung der Uhrzeit zu schreiben, um Zeitpläne für Programme aufzustellen. Wenn Sie ein Programm jedoch nur kurz anhalten wollen, nutzen Sie stattdessen `time.sleep()`. Anstatt die Zeitplanung des Betriebssystems zu nutzen, können Sie den Code auch eine Schleife ausführen lassen, bis ein bestimmter Zeitpunkt erreicht ist, und bei jedem Durchlauf `time.sleep(1)` aufrufen.

### Websites mit Python aufrufen

Statt `subprocess.Popen` können Sie auch die Funktion `webbrowser.open()` verwenden, um den Webbrowser von Ihrem Programm aus zu starten und darin eine bestimmte Website zu öffnen. Einzelheiten darüber erfahren Sie im Abschnitt »Projekt: mapIt.py mit dem Modul `webbrowser`« in Kapitel 11.

### Andere Python-Skripte ausführen

Von Python aus können Sie wie alle anderen Anwendungen auch andere Python-Skripte starten. Übergeben Sie dazu die ausführbare Datei `python.exe` und den

Namen des auszuführenden .py-Skripts an Popen(). Beispielsweise startet der folgende Code das Skript *hello.py* aus Kapitel 1:

```
>>> subprocess.Popen(['C:\\python34\\python.exe', 'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

Sie müssen Popen() eine Liste übergeben, die den String des Pfads zu der ausführbaren Datei von Python und den String mit dem Dateinamen des auszuführenden Skripts enthält. Wenn das Skript Befehlszeilenargumente benötigt, fügen Sie sie hinter dem Skriptnamen als weitere Einträge in die Liste ein. Der Speicherort der ausführbaren Datei von Python ist *C:\python34\python.exe* (Windows), */Library/Frameworks/Python.framework/Versions/3.3/bin/python3* (OS X) bzw. */usr/bin/python3* (Linux).

Anders als beim Import eines Python-Programms als Modul laufen beim Aufruf eines Python-Programms aus einem anderen die beiden Programme in getrennten Prozessen und können nicht gegenseitig auf ihre Variablen zugreifen.

## Dateien in ihren Standardanwendungen öffnen

Wenn Sie auf einem Computer auf eine .txt-Datei doppelklicken, wird automatisch die Anwendung geöffnet, die mit dieser Endung verknüpft ist. Auf dem Rechner sind bereits viele solcher Verknüpfungen eingerichtet. Auf die gleiche Weise kann Python Dateien mithilfe von Popen() öffnen.

Jedes Betriebssystem verfügt über ein Programm, das das Gleiche bewirkt wie ein Doppelklick auf eine Dokumentendatei. Unter Windows ist dies das Programm *start*, unter OS X das Programm *open*, unter Ubuntu Linux *see*. Wenn Sie den folgenden Code in der interaktiven Shell ausprobieren, übergeben Sie Popen() je nachdem, welches Betriebssystem Sie verwenden, 'start', 'open' oder 'see':

```
>>> fileObj = open('hello.txt', 'w')
>>> fileObj.write('Hello world!')
12
>>> fileObj.close()
>>> import subprocess
>>> subprocess.Popen(['start', 'hello.txt'], shell=True)
```

Hier schreiben wir Hello world! in eine neue Datei namens *hello.txt*. Anschließend rufen wir die Funktion Popen() auf und übergeben ihr eine Liste, die den Programmnamen (hier 'start' für Windows) und den Dateinamen enthält. Zusätzlich übergeben wir noch das Schlüsselwortargument *shell=True*, das aber nur auf Windows erforderlich ist. Da das Betriebssystem alle Dateiverknüpfungen kennt,

kann es herausfinden, dass es zum Öffnen der Datei *hello.txt* beispielsweise das Programm *Notepad.exe* starten muss.

Unter OS X wird `open` sowohl zum Öffnen von Dokumentendateien als auch von Programmen verwendet. Auf einem Mac können Sie Folgendes in die interaktive Shell eingeben:

```
>>> subprocess.Popen(['open', '/Applications/Calculator.app/'])
<subprocess.Popen object at 0x10202ff98>
```

Dadurch wird die Taschenrechneranwendung geöffnet.

### Die Unix-Philosophie

Gut entworfene Programme, die von anderen Programmen geöffnet werden können, bieten viel mehr Möglichkeiten als ihr Code an sich. Die *Unix-Philosophie* ist ein Satz von Softwaredesignprinzipien, die von den Programmierern des Betriebssystems Unix aufgestellt wurden (auf dem die modernen Betriebssysteme Linux und OS X beruhen). Sie besagt, dass es besser ist, kleine Programme zu schreiben, die auf einen bestimmten Zweck beschränkt sind und miteinander zusammenarbeiten können, als eine riesige, funktionsreiche Anwendung. Die kleinen Programme lassen sich einfacher verstehen und können dank ihrer Interoperabilität die Bausteine leistungsfähigerer Anwendungen bilden.

Smartphone-Apps folgen ebenfalls dieser Philosophie. Bei einer Restaurant-App, die Ihnen den Weg zum nächsten Café anzeigen soll, haben die Entwickler nicht versucht, das Rad neu zu erfinden, indem sie ihren eigenen Code zur Kartenanzeige geschrieben haben. Stattdessen startet die Restaurant-App einfach eine Karten-App und übergibt ihr die Adresse des Cafés, ebenso wie Python eine Funktion aufruft und ihr Argumente übergibt.

Die Python-Programme, die Sie beim Durcharbeiten dieses Buchs bis jetzt geschrieben haben, folgen in den meisten Fällen der Unix-Philosophie, insbesondere in einem sehr wichtigen Punkt: Sie verwenden Befehlszeilenargumente, anstatt die Funktion `input()` aufzurufen. Wenn alle Informationen, die das Programm braucht, vorab bereitgestellt werden können, ist es besser, sie gleich als Befehlszeilenargumente zu übergeben, anstatt darauf zu warten, dass der Benutzer sie eingibt. Dabei können die Befehlszeilenargumente sowohl von einem Menschen als auch von einem anderen Programm bereitgestellt werden. Durch diese interoperable Vorgehensweise kann Ihr Programm auch von einem anderen Programm genutzt werden.

Die einzige Ausnahme bilden Passwörter, die nicht als Befehlszeilenargumente übergeben werden sollten, da sie dabei im Befehlsverlauf aufgezeichnet werden könnten. Stattdessen sollte ein Programm `input()` aufrufen, wenn der Benutzer ein Passwort eingeben muss.

Mehr über die Unix-Philosophie erfahren Sie auf <https://de.wikipedia.org/wiki/Unix-Philosophie>.

## Projekt: Ein einfaches Countdown-Programm

Ein einfaches Countdown-Programm lässt sich genauso schwer finden wie ein einfaches Stoppuhrprogramm. Daher schreiben wir selbst eines, das nach dem Ende des Countdowns einen Alarm auslöst.

Dieses Programm soll Folgendes tun:

- Von 60 abwärts zählen
- Eine Klangdatei (*alarm.wav*) abspielen, wenn der Countdown bei 0 angekommen ist

Dazu muss der Code folgende Aufgaben ausführen:

- Zwischen der Anzeige der einzelnen Zahlen im Countdown jeweils `time.sleep()` aufrufen, um eine Sekunde lang anzuhalten
- Mit `subprocess.Popen()` die Klangdatei in ihrer Standardanwendung öffnen

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das noch leere Programm als *countdown.py*.

### Schritt 1: Der Countdown

Dieses Programm benötigt das Modul `time` für die Funktion `time.sleep()` und das Modul `subprocess` für `subprocess.Popen()`. Geben Sie folgenden Code in *countdown.py* ein:

```
#! python3
# countdown.py - Ein einfaches Countdown-Skript

import time, subprocess

timeLeft = 60    ❶
while timeLeft > 0:
    print(timeLeft, end='') ❷
    time.sleep(1) ❸
    timeLeft = timeLeft - 1 ❹

# TODO: Am Ende des Countdowns eine Klangdatei abspielen
```

Nach dem Import von `time` und `subprocess` erstellen Sie die Variable `timeLeft`, die festhält, wie viele Sekunden im Countdown noch verbleiben (❶). Der Anfangswert beträgt hier 60, aber Sie können ihn auf jeden Wert einstellen, den Sie brauchen. Es ist auch möglich, diesen Wert über ein Befehlszeilenargument bereitzustellen.

In einer `while`-Schleife zeigen Sie dann die verbleibende Anzahl an Sekunden an (❷), halten eine Sekunde lang an (❸) und verringern dann den Wert von

timeLeft um 1 (❸). Danach beginnt die Schleife von vorn. Sie wird so lange durchlaufen, wie der Wert von timeLeft größer als 0 ist. Danach ist der Countdown beendet.

## Schritt 2: Die Klangdatei abspielen

Es gibt Drittanbietermodule, die Klangdateien verschiedener Formate abspielen können, doch am einfachsten ist es, einfach die Anwendung zu starten, die ohnehin schon für diesen Zweck vorhanden ist. Das Betriebssystem kann an der Dateierweiterung *.wav* erkennen, welche Anwendung es starten soll, um die Datei abzuspielen. Statt einer *.wav*-Datei können Sie hier auch eine Klangdatei in einem anderen Format verwenden, z. B. *.mp3* oder *.ogg*.

Die Datei *alarm.wav* können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen, Sie können aber auch eine beliebige andere Klangdatei verwenden, die sich auf Ihrem Computer befindet.

Ergänzen Sie den Code wie folgt:

```
#! python3
# countdown.py - Ein einfaches Countdown-Skript

import time, subprocess

-- schnipp --

# Spielt am Ende des Countdowns eine Klangdatei ab
subprocess.Popen(['start', 'alarm.wav'], shell=True)
```

Nach dem Ende der while-Schleife wird *alarm.wav* (oder eine andere Klangdatei Ihrer Wahl) abgespielt, um den Benutzer zu benachrichtigen, dass der Countdown abgelaufen ist. Unter Windows muss die Liste, die Sie an Popen() übergeben, sowohl 'start' als auch das Schlüsselwertargument *shell=True* enthalten. Unter OS X verwenden Sie 'open' statt 'start' und verzichten auf *shell=True*.

Sie können auch eine Textdatei mit der Meldung *Break time is over!* speichern und dann am Ende des Countdowns mithilfe von Popen() öffnen, anstatt die Klangdatei abzuspielen. Dadurch wird im Grunde genommen ein Popup-Fenster mit einer Meldung geöffnet. Es ist auch möglich, am Ende des Countdowns mit *webbrowser.open()* eine bestimmte Website zu öffnen. Anderes als bei manchen kostenlosen Countdown-Anwendungen, die es online gibt, können Sie den Alarm in Ihrem Countdown ganz nach Belieben gestalten.

## Vorschläge für ähnliche Programme

Ein Countdown kann für eine einfache Verzögerung sorgen, bevor die Ausführung eines Programms fortgesetzt wird. Das können Sie auch für andere Anwendungen und Funktionen nutzen:

- Verwenden Sie `time.sleep()`, um dem Benutzer die Möglichkeit zu geben, **Strg** + **C** zu drücken, um eine Aktion abzubrechen, z. B. das Löschen von Dateien. Das Programm kann eine Meldung ausgeben wie: »Drücken Sie **Strg** + **C**, um den Vorgang abzubrechen.« Die KeyboardInterrupt-Ausnahmen, die durch die Betätigung dieser Tasten ausgelöst werden, behandeln Sie dann mit `try-` und `except-`Anweisungen.
- Für einen längeren Countdown können Sie `timedelta`-Objekte verwenden, um die Anzahl der Tage, Stunden, Minuten und Sekunden bis zu einem bestimmten zukünftigen Zeitpunkt (Geburtstag? Hochzeitstag?) zu messen.

## Zusammenfassung

Die Unix-Epoche (0 Uhr am 1. Januar 1970 UTC) ist ein Standardreferenzzeitpunkt für viele Programmiersprachen, darunter auch Python. Die Funktion `time.time()` gibt einen Epochenzzeitstempel zurück, also einen Fließkommawert mit der Anzahl der Sekunden, die seit der Unix-Epoche verstrichen sind. Für Kalenderarithmetik und zum Formatieren oder Analysieren von Strings mit Datumsinformationen ist jedoch das Modul `datetime` besser geeignet.

Die Funktion `time.sleep()` blockiert die Ausführung für die (in Sekunden) angegebene Dauer (d. h., sie gibt so lange die Steuerung nicht zurück). Damit können Sie in einem Programm Pausen einlegen. Wenn Sie dafür sorgen wollen, dass Ihr Programm zu einem bestimmten Zeitpunkt startet, können Sie jedoch die Zeitplanungswerzeuge des Betriebssystems verwenden. Anleitungen dazu erhalten Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).

Das Modul `threading` dient dazu, mehrere Threads zu erstellen, was sehr praktisch ist, wenn Sie mehrere Dateien herunterladen oder andere Aufgaben gleichzeitig ausführen wollen. Sorgen Sie aber dafür, dass die Threads nur lokale Variablen lesen und schreiben, da es sonst zu Nebenläufigkeitsproblemen kommen kann.

Mit der Funktion `subprocess.Popen()` können Python-Programme auch andere Anwendungen starten. Um in diesen Anwendungen gleich bestimmte Dokumente zu öffnen, können Sie `Popen()` entsprechende Befehlszeilenargumente übergeben. Alternativ können Sie mit `Popen()` das Programm `start`, `open` oder `see` öffnen, um die Dateiverknüpfungen des Computers zu nutzen und ein Dokument automatisch in der richtigen Anwendung zu öffnen. Durch die Nutzung anderer auf Ihrem

Computer vorhandenen Anwendungen können Ihre Python-Programme deren Möglichkeiten nutzen, um ihre Aufgaben zu erfüllen.

## Wiederholungsfragen

1. Was ist die Unix-Epoche?
2. Welche Funktion gibt die Anzahl der Sekunden zurück, die seit der Unix-Epoche verstrichen sind?
3. Wie können Sie ein Programm genau fünf Sekunden lang anhalten lassen?
4. Was gibt die Funktion `round()` zurück?
5. Was ist der Unterschied zwischen einem `datetime`- und einem `timedelta`-Objekt?
6. Wie können Sie eine Funktion namens `spam()` aufrufen und ihren Code in einem eigenen Thread ausführen?
7. Was sollten Sie tun, um in Programmen mit mehreren Threads Nebenläufigkeitsprobleme zu vermeiden?
8. Wie sorgen Sie dafür, dass ein Python-Programm das Programm `calc.exe` im Ordner `C:\Windows\System32` ausführt?

## Übungsprojekte

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben erfüllen:

### Elegantere Stoppuhr

Erweitern Sie das Stoppuhrprojekt aus diesem Kapitel, um die Ausgabe mit den Stringmethoden `rjust()` und `ljust()` aufzuhübschen. (Diese Methoden wurden in Kapitel 6 besprochen.) Bisher sieht die Ausgabe wie folgt aus:

```
Lap #1: 3.56 (3.56)
Lap #2: 8.63 (5.07)
Lap #3: 17.68 (9.05)
Lap #4: 19.11 (1.43)
```

Daraus soll nun Folgendes werden:

```
Lap # 1:    3.56 (  3.56)
Lap # 2:    8.63 (  5.07)
Lap # 3:   17.68 (  9.05)
Lap # 4:   19.11 (  1.43)
```

Beachten Sie, dass es sich bei `lapNum`, `lapTime` und `totalTime` um Integer- und Fließkommavariablen handelt. Um die Stringmethoden für sie aufzurufen, müssen Sie ihre Stringversionen erstellen.

Kopieren Sie die Textausgabe anschließend mit dem in Kapitel 6 eingeführten Modul `pyperclip` in die Zwischenablage, sodass der Benutzer sie danach in eine Textdatei oder eine E-Mail einfügen kann.

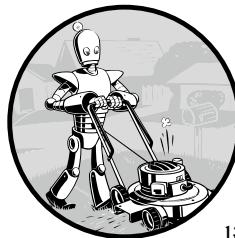
### **Webcomic-Download-Programm mit Zeitplanung**

Schreiben Sie ein Programm, das die Websites verschiedener Webcomics überprüft und automatisch die neuen Bilder herunterlädt, wenn es seit dem letzten Besuch des Programms Aktualisierungen gegeben hat. Mit dem Zeitplanungswerkzeug Ihres Betriebssystems (Taskplaner in Windows, `launchd` in OS X und `cron` in Linux) können Sie das Python-Programm einmal täglich ausführen lassen. Das Programm soll dann die neuen Comics herunterladen und auf Ihren Desktop kopieren, sodass Sie ihn dort leicht finden können. Dadurch müssen Sie nicht mehr selbst nachsehen, ob es neues Material auf diesen Websites gibt. (Eine Liste von Webcomics finden Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).)



# 16

## E-Mails und Textnachrichten senden



Den E-Mail-Posteingang zu überprüfen und auf E-Mails zu antworten, kostet täglich viel Zeit. Natürlich kann man nicht einfach ein Programm schreiben, das sich darum kümmert, da Sie ja schließlich Antworten auf die eingegangenen E-Mails formulieren müssen.

Allerdings lassen sich viele Aufgaben im Zusammenhang mit E-Mails automatisieren.

Nehmen wir beispielsweise an, Sie haben ein Arbeitsblatt mit Kundenadressen und möchten diesen Kunden je nach Alter und Wohnort unterschiedliche Formbriefe schicken. Mit kommerzieller Software mag das nicht genauso möglich sein, wie Sie es sich vorstellen, aber zum Glück können Sie ein eigenes Programm dafür schreiben und sich damit die Zeit und Mühe ersparen, Textpassagen in solchen Form-E-Mails hin und her zu kopieren.

Es ist auch möglich, Programme zu schreiben, die E-Mails oder SMS senden, um Sie über irgendwelche Vorgänge zu informieren, während Sie nicht am Computer sitzen. Wenn Sie eine Aufgabe automatisieren, die mehrere Stunden in Anspruch nimmt, dann wollen Sie nicht alle paar Minuten nachsehen gehen, wie weit das Programm gekommen ist. Stattdessen kann Ihnen das Programm einfach eine

SMS an Ihr Handy schicken, wenn es fertig ist. Dadurch können Sie sich in der Zwischenzeit auf andere Dinge konzentrieren.

## SMTP

Um Webseiten über das Internet zu schicken, stützen sich Computer auf das Protokoll HTTP. Zum Senden von E-Mails dagegen greifen sie auf SMTP (Simple Mail Transfer Protocol) zurück. Dieses Protokoll schreibt vor, wie E-Mail-Nachrichten formatiert, verschlüsselt und von einem Mailserver zum anderen weitergeleitet werden müssen, und legt alle Einzelheiten der Vorgänge fest, die ablaufen, nachdem Sie auf *Senden* geklickt haben. Diese technischen Einzelheiten müssen Sie jedoch nicht kennen, da das Python-Modul `smtplib` einige einfache Funktionen dafür bereitstellt. SMTP regelt nur das Senden von E-Mails. Für den Empfang ist das Protokoll IMAP verantwortlich, das in einem eigenen Abschnitt weiter hinten in diesem Kapitel beschrieben wird.

## E-Mails senden

Leider bietet Python keine ansprechende grafische Oberfläche zum Senden von E-Mails wie Outlook, Thunderbird oder Websites wie Gmail oder Yahoo! Mail. Um die einzelnen SMTP-Vorgänge auszuführen, müssen Sie stattdessen jeweils Funktionen aufrufen, wie das folgende Beispiel für die interaktive Shell zeigt.

### Hinweis

Der folgende Code dient nur dazu, das Senden von E-Mails mit Python zu veranschaulichen. Sie können ihn jedoch nicht erfolgreich in IDLE ausprobieren, da die verwendeten Adressen `smtplib.example.com`, `bob@example.com` und `alice@example.com` sowie `MY_SECRET_PASSWORD` nur fiktive Beispielwerte sind.

```
>>> import smtplib
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE 35882577\
n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
>>> smtpObj.login('bob@example.com', 'MY_SECRET_PASSWORD')
(235, b'2.7.0 Accepted')
>>> smtpObj.sendmail('bob@example.com', 'alice@example.com', 'Subject: So
long.\nDear Alice, so long and thanks for all the fish. Sincerely, Bob')
{}
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

In den folgenden Abschnitten gehen wir dieses Beispiel Schritt für Schritt durch. Dabei ersetzen Sie die Beispielwerte auch durch reale Angaben, um sich an einem SMTP-Server anzumelden, eine E-Mail zu senden und die Verbindung zum Server wieder zu trennen.

### Verbindung mit einem SMTP-Server aufnehmen

Wenn Sie schon einmal in Thunderbird, Outlook oder einem vergleichbaren Programm Ihr E-Mail-Konto eingerichtet haben, wissen Sie bereits, wie Sie den SMTP-Server und -Port angeben müssen. Diese Einstellungen sind bei jedem E-Mail-Provider anders, aber Sie können im Web nach dem Providernamen und »SMTP-Einstellungen« oder »SMTP settings« suchen, um sie herauszufinden.

Der Domänenname des SMTP-Servers ist gewöhnlich der Domänenname des Providers mit vorangestelltem *smtp*., beispielsweise *smtp.gmail.com* für Gmail. Tabelle 16–1 gibt eine Übersicht über die SMTP-Server einiger gebräuchlicher E-Mail-Provider. (Der Port ist ein Integerwert. Meistens ist es Port 587, der vom Befehlsverschlüsselungsstandard TLS genutzt wird.)

Provider	Domänenname des SMTP-Servers
Gmail	<i>smtp.gmail.com</i>
Outlook.com/Hotmail.com	<i>smtp-mail.outlook.com</i>
Yahoo Mail	<i>smtp.mail.yahoo.com</i>
AT&T	<i>smpt.mail.att.net</i> (Port 465)
Comcast	<i>smtp.comcast.net</i>
Verizon	<i>smtp.verizon.net</i> (Port 465)

**Tab. 16–1**    SMTP-Server bekannter E-Mail-Provider

Mit diesen Informationen können Sie ein SMTP-Objekt erstellen, indem Sie die Funktion `smtplib.SMTP()` aufrufen und ihr den Domänennamen als String- und die Portnummer als Integerargument übergeben. Das SMTP-Objekt steht für eine Verbindung zu einem SMTP-Mailserver und verfügt über Methoden zum Senden von E-Mails. Beispielsweise erstellt der folgende Aufruf ein SMTP-Objekt für die Verbindung zu Gmail:

```
>>> smtpObj = smtplib.SMTP('smtp.gmail.com', 587)
>>> type(smtpObj)
<class 'smtplib.SMTP'>
```

Die Ausgabe von `type(smtpObj)` zeigt, dass in `smtpObj` ein SMTP-Objekt gespeichert ist. Dieses Objekt brauchen Sie, um die Methoden zum Anmelden und zum Senden von E-Mails aufrufen zu können. Schlägt der Aufruf von `smtplib.SMTP()` fehl, kann das daran liegen, dass der SMTP-Server TLS an Port 587 nicht unterstützt. In diesem Fall müssen Sie das SMTP-Objekt stattdessen mit `smtplib.SMTP_SSL()` und Port 465 erstellen:

```
>>> smtpObj = smtplib.SMTP_SSL('smtp.gmail.com', 465)
```

### Hinweis

Wenn Sie nicht mit dem Internet verbunden sind, löst Python `socket.gaierror: [Errno 11004] getaddrinfo failed` oder eine ähnliche Ausnahme aus.

Für Ihre Programme sind die Unterschiede zwischen TLS und SSL nicht von Belang. Sie müssen nur wissen, welchen Verschlüsselungsstandard Ihr SMTP-Server nutzt, damit Sie auf die richtige Weise Verbindung mit ihm aufnehmen können. In allen nachfolgenden Beispielen enthält die Variable `smtpObj` ein MTP-Objekt, das von der Funktion `smtplib.SMTP()` oder `smtplib.SMTP_SSL()` zurückgegeben wurde.

### Die »Hallo«-Nachricht an den SMTP-Server senden

Wenn Sie das SMTP-Objekt haben, rufen Sie dessen etwas eigenartig benannte Methode `ehlo()` auf, um dem SMTP-Mailserver »Hallo« zu sagen. Dieser Gruß ist der erste Schritt bei der SMTP-Kommunikation und wichtig, um eine Verbindung zu dem Server herzustellen. Die Einzelheiten des Protokolls müssen Sie dazu jedoch nicht kennen. Wichtig ist nur, dass Sie nach dem Abruf des SMTP-Objekts als Erstes `ehlo()` aufrufen. Wenn Sie das versäumen, werden alle weiteren Methodenaufrufe zu Fehlern führen. Das folgende Beispiel zeigt den Aufruf von `ehlo()` und den Rückgabewert:

```
>>> smtpObj.ehlo()
(250, b'mx.google.com at your service, [216.172.148.131]\nSIZE 35882577\
n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
```

Wenn das erste Element in dem zurückgegebenen Tupel der Integerwert 250 ist (der Code für »Erfolg« in SMTP), dann war die Begrüßung erfolgreich.

## Die TLS-Verschlüsselung einleiten

Wenn die Verbindung mit dem SMTP-Server über Port 587 läuft (wenn Sie also die TLS-Verschlüsselung nutzen), müssen Sie als Nächstes die Methode `starttls()` aufrufen. Dieser erforderliche Schritt aktiviert die Verschlüsselung für die Verbindung. Erfolgt die Kommunikation dagegen über Port 465 (mit SSL), ist die Verschlüsselung bereits eingerichtet, sodass Sie diesen Schritt nicht ausführen.

Das folgende Beispiel zeigt den Aufruf der Methode `starttls()`:

```
>>> smtp0bj.starttls()  
(220, b'2.0.0 Ready to start TLS')
```

`starttls()` versetzt die SMTP-Verbindung in den TLS-Modus. Die Angabe 220 im Rückgabewert teilt Ihnen mit, dass der Server bereit ist.

## Am SMTP-Server anmelden

Nachdem die verschlüsselte Verbindung zum SMTP-Server eingerichtet ist, können Sie sich mit Ihrem Benutzernamen (gewöhnlich Ihrer E-Mail-Adresse) und Ihrem E-Mail-Passwort anmelden. Dazu rufen Sie die Methode `login()` auf:

```
>>> smtp0bj.login('my_email_address@gmail.com', 'MY_SECRET_PASSWORD')  
(235, b'2.7.0 Accepted')
```

### Anwendungsspezifische Passwörter in Gmail

Gmail verfügt über ein zusätzliches Sicherheitsmerkmal für Google-Konten, nämlich die sogenannten *anwendungsspezifischen Passwörter*. Wenn Ihr Programm versucht, sich anzumelden, Sie dabei aber die Fehlermeldung `Application-specific password required` erhalten, müssen Sie ein solches Passwort für Ihr Python-Skript einrichten. Genaue Anleitungen dazu erhalten Sie in den Ressourcen auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).

Als erstes Argument übergeben Sie den String Ihrer E-Mail-Adresse, als zweites den String Ihres Passworts. Der Rückgabewert 235 bedeutet, dass die Authentifizierung erfolgreich verlaufen ist. Bei Angabe eines falschen Passworts löst Python die Ausnahme `smtplib.SMTPAuthenticationError` aus.

## Warnung

Passwörter im Quellcode anzugeben, ist gefährlich, denn wenn irgendjemand das Programm kopiert, hat er dadurch Zugriff auf Ihr E-Mail-Konto! Es ist besser, `input()` aufzurufen, damit der Benutzer sein Passwort eingeben kann. Es mag zwar unbequem sein, bei jeder Ausführung des Programms sein Passwort einzugeben zu müssen, aber dadurch verhindern Sie, dass das Passwort in einer ungeschützten Datei auf Ihrem Computer steht, wo ein Hacker oder jemand, der Ihren Laptop stiehlt, ganz leicht darauf zugreifen kann.

## Eine E-Mail senden

Nachdem Sie am SMTP-Server Ihres Providers angemeldet sind, können Sie die Methode `sendmail()` aufrufen, um die E-Mail zu verschicken:

```
>>> smtpObj.sendmail('my_email_address@gmail.com', 'recipient@example.com',
'Subject: So long.\nDear Alice, so long and thanks for all the fish.
Sincerely, Bob')
{}
```

Die Methode `sendmail()` braucht drei Argumente:

- Ihre E-Mail-Adresse als String (als Absenderadresse für das Feld *Von*)
- Die E-Mail-Adresse des Empfängers als String (für das Feld *An*); bei mehreren Empfängern geben Sie eine Liste von Strings an.
- Den Rumpf der E-Mail als String

Der String für den E-Mail-Rumpf *muss* mit 'Subject: \n' für die Betreffzeile beginnen. Das Zeilenumbruchzeichen '\n' trennt diese Zeile vom Haupttext der E-Mail.

Der Rückgabewert von `sendmail()` ist ein Dictionary mit einem Schlüssel-Wert-Paar für jeden Empfänger, für den die Zustellung *fehlgeschlagen* ist. Ein leeres Dictionary bedeutet, dass die E-Mail erfolgreich an alle Empfänger zugestellt wurde.

## Die Verbindung zum SMTP-Server trennen

Wenn Sie das Senden der E-Mails abgeschlossen haben, müssen Sie die Methode `quit()` aufrufen, um die Verbindung Ihres Programms zum SMTP-Server zu trennen:

```
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pb.52 - gsmtp')
```

Der Integer 211 im Rückgabewert bedeutet, dass die Sitzung beendet ist.

## IMAP

Das Gegenstück zu SMTP, dem Protokoll zum Senden von E-Mails, ist IMAP (Internet Message Access Protocol), das die Kommunikation mit dem Server des E-Mail-Providers beschreibt, um die an Ihre Adresse gesendeten E-Mails abzurufen. Zwar ist im Lieferumfang von Python das Modul `imaplib` enthalten, allerdings lässt sich das Drittanbietermodul `imapclient` viel einfacher verwenden. Dieses Kapitel gibt eine Einführung in `imapclient`. Die vollständige Dokumentation finden Sie auf <http://imapclient.readthedocs.org/>.

Das Modul `imapclient` lädt E-Mails von einem IMAP-Server herunter, allerdings in einem ziemlich komplizierten Format, sodass Sie sie erst in einfache Stringwerte umwandeln müssen. Die Schwerarbeit dabei nimmt Ihnen das Modul `pyzmail` ab, dessen vollständige Dokumentation Sie auf <http://www.magiksys.net/pyzmail/> finden.

Installieren Sie `imapclient` und `pyzmail` in einem Terminal-Fenster. Eine Anleitung zur Installation von Drittanbietermodulen finden Sie in Anhang A.

## E-Mails mit IMAP abrufen und löschen

Eine E-Mail zu finden und abzurufen, erfordert in Python mehrere Schritte und die beiden Module `imapclient` und `pyzmail`. Das folgende Beispiel gibt einen Überblick über den Gesamtvorgang. Dabei melden Sie sich am IMAP-Server an, suchen nach E-Mails, rufen sie ab und entnehmen ihnen dann den Text:

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
>>> imapObj.login('my_email_address@gmail.com', 'MY_SECRET_PASSWORD')
'my_email_address@gmail.com Jane Doe authenticated (Success)'
>>> imapObj.select_folder('INBOX', readonly=True)
>>> UIDs = imapObj.search(['SINCE 05-Jul-2014'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
>>> rawMessages = imapObj.fetch([40041], ['BODY[]', 'FLAGS'])
>>> import pyzmail
>>> message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'jdoe@example.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
```

```
>>> message.text_part != None
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
'Follow the money.\r\n\r\nEd\r\n'
>>> message.html_part != None
True
>>> message.html_part.get_payload().decode(message.html_part.charset)
'<div dir="ltr"><div>So long, and thanks for all the fish!<br></div>-A1<br></div>\r\n'
>>> imapObj.logout()
```

Diese Schritte müssen Sie nicht auswendig lernen. Wir werden sie im Folgenden einen nach dem anderen durchgehen und wenn Sie Ihr Gedächtnis wieder auffrischen müssen, können Sie sich einfach diese Übersicht erneut ansehen.

### Verbindung mit einem IMAP-Server aufnehmen

Um Verbindung mit einem SMTP-Server aufzunehmen und E-Mails senden zu können, brauchen Sie ein SMTP-Objekt. Das Gegenstück dazu ist ein `IMAPClient`-Objekt, das Sie benötigen, um Verbindung mit einem IMAP-Server aufzunehmen und E-Mails empfangen zu können. Dazu müssen Sie den Domänennamen für den IMAP-Server Ihres E-Mail-Providers kennen, der nicht mit dem Domänennamen des SMTP-Servers identisch ist. Tabelle 16–2 nennt die IMAP-Server mehrerer beliebter Provider.

Provider	Domänenname des IMAP-Servers
Gmail	<code>imap.gmail.com</code>
Outlook.com/Hotmail.com	<code>imap-mail.outlook.com</code>
Yahoo Mail	<code>imap.mail.yahoo.com</code>
AT&T	<code>imap.mail.att.net</code>
Comcast	<code>imap.comcast.net</code>
Verizon	<code>incoming.verizon.net</code>

**Tab. 16–2** IMAP-Server bekannter E-Mail-Provider

Mit diesen Informationen können Sie die Funktion `imapclient.IMAPClient()` aufrufen, um ein `IMAPClient`-Objekt zu erstellen. Für die meisten E-Mail-Provider ist eine SSL-Verschlüsselung erforderlich, sodass Sie dafür das Schlüsselwortargument `ssl=True` übergeben müssen. Geben Sie Folgendes in die interaktive Shell ein, wobei Sie jedoch den Domänennamen Ihres Providers angeben müssen:

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
```

In allen Beispielen der nachfolgenden Abschnitte enthält die Variable `imap0bj` ein `IMAPClient`-Objekt, das von der Funktion `imapclient.IMAPClient()` zurückgegeben wurde. In diesem Zusammenhang ist der *Client* das Objekt, das die Verbindung mit dem Server aufnimmt.

### Am IMAP-Server anmelden

Rufen Sie nun die Methode `login()` des `IMAPClient`-Objekts auf und übergeben Sie ihm den Benutzernamen (gewöhnlich Ihre E-Mail-Adresse) und Ihr Passwort als Strings:

```
>>> imap0bj.login('my_email_address@gmail.com', 'MY_SECRET_PASSWORD')
'my_email_address@gmail.com Jane Doe authenticated (Success)'
```

### Warnung

Denken Sie daran, die Passwörter nicht unmittelbar in den Code zu schreiben! Gestalten Sie Ihr Programm stattdessen so, dass es ein von `input()` zurückgegebenes Passwort annimmt.

Wenn der IMAP-Server die angegebene Kombination aus Benutzername und Passwort ablehnt, löst Python die Ausnahme `imaplib.error` aus. Für Gmail-Konten müssen Sie unter Umständen ein anwendungsspezifisches Passwort angeben. Mehr darüber erfahren Sie in dem Kasten »Anwendungsspezifische Passwörter in Gmail« weiter vorn in diesem Kapitel.

### Nach E-Mails suchen

Um nach der Anmeldung E-Mails abzurufen, müssen Sie zwei Dinge tun: Erstens müssen Sie einen Ordner auswählen, der durchsucht werden soll, und zweitens die Methode `search()` des `IMAPClient`-Objekts aufrufen und ihr einen String von IMAP-Suchwörtern übergeben.

### Einen Ordner auswählen

Praktisch jedes E-Mail-Konto verfügt standardmäßig über den Ordner `INBOX` für den Posteingang. Mit der Methode `list_folders()` des `IMAPClient`-Objekts können Sie sich jedoch auch eine Liste verfügbarer Ordner anzeigen lassen. Diese Liste besteht aus Tupeln, die jeweils Informationen über einen einzelnen Ordner enthalten. Setzen Sie das Beispiel in der interaktiven Shell wie folgt fort:

```
>>> import pprint
>>> pprint.pprint(imapObj.list_folders())
[('\'HasNoChildren\',), '/', 'Drafts'),
 ('\'HasNoChildren\',), '/', 'Filler'),
 ('\'HasNoChildren\',), '/', 'INBOX'),
 ('\'HasNoChildren\',), '/', 'Sent'),
 -- schnipp --
 (('__HasNoChildren', '__Flagged'), '/', '[Gmail]/Starred'),
 (('__HasNoChildren', '__Trash'), '/', '[Gmail]/Trash')]
```

So ungefähr kann die Ausgabe bei einem Gmail-Konto aussehen. (Bei Gmail werden die Ordner als *Labels* bezeichnet, sie funktionieren aber genauso wie Ordner.) Die drei Werte in den Tupeln, also beispielsweise `(('\HasNoChildren',), '/', 'INBOX')`, haben folgende Bedeutung:

- Ein Tupel mit den Flags des Ordners. (Eine Erklärung, was diese Flags bedeuten, würde hier zu weit führen, Sie können sie aber getrost ignorieren.)
- Das Trennzeichen, das im Namensstring zwischen Eltern- und Kindordnern verwendet wird
- Der vollständige Name des Ordners

Um den zu durchsuchenden Ordner auszuwählen, übergeben Sie seinen Namen als String an die Methode `select_folder()` des `IMAPClient`-Objekts.

```
>>> imapObj.select_folder('INBOX', readonly=True)
```

Um den Rückgabewert von `select_folder()` brauchen Sie sich nicht zu kümmern. Wenn es den ausgewählten Ordner nicht gibt, löst Python die Ausnahme `imaplib.error` aus.

Das Schlüsselwortargument `readonly=True` verhindert, dass Sie die E-Mails in dem Ordner mit den nachfolgenden Methoden aufrufen versehentlich ändern oder löschen. Sofern Sie nicht ausdrücklich vorhaben, E-Mails zu löschen, sollten Sie `readonly` immer auf `True` setzen.

## Die Suche durchführen

Mit der Methode `search()` des `IMAPClient`-Objekts können Sie nun in dem ausgewählten Ordner nach E-Mails suchen. Als Argument übergeben Sie eine Liste von Strings mit den Suchschlüsseln von IMAP, die in Tabelle 16–3 beschrieben werden.

Suchschlüssel	Bedeutung
'ALL'	Gibt alle Nachrichten im Ordner zurück. Wenn Sie sämtliche Nachrichten in einem umfangreichen Ordner abrufen, kann das zu Problemen aufgrund der Größeneinschränkungen in <code>imaplib</code> führen. Mehr darüber erfahren Sie im Abschnitt »Größeneinschränkungen« weiter hinten in diesem Kapitel.
'BEFORE <i>Datum'</i> 'ON <i>Datum'</i> 'SINCE <i>Datum'</i>	Mit diesen drei Suchschlüsseln geben Sie die Nachrichten zurück, die der IMAP-Server vor, am oder nach dem angegebenen Datum empfangen hat. Das Datum muss nach dem Muster 05-Jul-2015 formatiert sein. Bei der Angabe 'SINCE 05-Jul-2015' erhalten Sie Nachrichten, die am und nach dem 5. Juli 2015 eingegangen sind, bei 'BEFORE 05-Jul-2015' jedoch nur Nachrichten, die vor dem 5. Juli eingetroffen sind, aber nicht am 5. Juli selbst.
'SUBJECT <i>String'</i> 'BODY <i>String'</i> 'TEXT <i>String'</i>	Gibt Nachrichten zurück, bei denen der angegebene String in der Betreffzeile, im Rumpf der Nachricht oder in beiden vorhanden ist. Wenn der String Leerzeichen enthält, müssen Sie ihn in doppelte Anführungszeichen einschließen: 'TEXT "search with spaces"'.
'FROM <i>String'</i> 'TO <i>String'</i> 'CC <i>String'</i> 'BCC <i>String'</i>	Gibt alle Nachrichten zurück, bei denen der angegebene String in der Absenderadresse (Von), der Empfängeradresse (An), der Kopieadresse (CC) bzw. der Blindkopieadresse (BCC) vorhanden ist. Wenn Sie in dem String mehrere Adressen angeben möchten, müssen Sie sie durch Leerzeichen trennen und den Gesamtstring in doppelte Anführungszeichen setzen: 'CC "firstcc@example.com secondcc@example.com"'.
'SEEN' 'UNSEEN'	Gibt alle Nachrichten mit bzw. ohne das Flag \Seen zurück. Dieses Flag erhält eine E-Mail, wenn Sie mit der (weiter hinten beschriebenen) Methode <code>fetch()</code> darauf zugreifen oder in einem E-Mail-Programm oder Webbrowser darauf klicken. Man spricht zwar eher davon, dass eine Nachricht »gelesen« wurde statt »gesehen«, aber in diesem Flag und dem Schlüssel wird trotzdem das Wort <i>seen</i> für denselben Vorgang verwendet.
'ANSWERED' 'UNANSWERED'	Gibt alle Nachrichten mit bzw. ohne das Flag \Answered zurück. Dieses Flag erhält eine E-Mail, wenn Sie darauf antworten.
'DELETED' 'UNDELETED'	Gibt alle Nachrichten mit bzw. ohne das Flag \Deleted zurück. Wenn Sie eine E-Mail mit der Methode <code>delete_messages()</code> löschen, erhält sie das Flag \Deleted, wird aber nicht endgültig entfernt. Das geschieht erst, wenn Sie die Methode <code>expunge()</code> dafür aufrufen (siehe »E-Mails löschen« weiter hinten in diesem Kapitel). Manche E-Mail-Provider führen jedoch automatisch eine endgültige Löschung durch, z. B. Gmail.
'DRAFT' 'UNDRAFT'	Gibt alle Nachrichten mit bzw. ohne das Flag \Draft zurück. Solche Entwürfe stehen jedoch meistens in einem eigenen Ordner namens <i>Drafts</i> und nicht im Posteingang.

Suchschlüssel	Bedeutung
'FLAGGED' 'UNFLAGGED'	Gibt alle Nachrichten mit bzw. ohne das Flag \Flagged zurück. Dieses Flag wird gewöhnlich gesetzt, wenn eine E-Mail als »Wichtig« oder »Dringend« gekennzeichnet wird.
'LARGER N' 'SMALLER N'	Gibt alle Nachrichten zurück, die größer bzw. kleiner als <i>N</i> Byte sind.
'NOT Suchschlüssel'	Gibt alle Nachrichten zurück, die der angegebene Suchschlüssel <i>nicht</i> zurückgeben würde.
'OR Suchschlüssel1 Suchschlüssel2'	Gibt alle Nachrichten zurück, die mit dem ersten oder dem zweiten Suchschlüssel übereinstimmen.

**Tab. 16–3 IMAP-Suchschlüssel**

Beachten Sie, dass Flags und Suchschlüssel von einigen IMAP-Servern anders gehandhabt werden können. Manchmal ist ein wenig Experimentieren in der interaktiven Shell notwendig, um herauszufinden, wie sich ein bestimmter Server genau verhält.

Im Listenargument der Methode `search()` können Sie auch mehrere IMAP-Suchschlüsselstrings übergeben. Zurückgegeben werden dann die Nachrichten, die *alle* Suchschlüssel erfüllen. Wollen Sie dagegen die Nachrichten haben, die mit beliebigen der Suchschlüssel übereinstimmen, verwenden Sie `OR`. Wenn Sie `NOT` oder `OR` verwenden, müssen Sie dahinter einen bzw. zwei vollständige Suchschlüssel angeben.

Die folgende Aufstellung zeigt einige Beispieldaufrufe der Methode `search()` samt einer Erklärung:

- `imapObj.search(['ALL'])` Gibt alle Nachrichten im zurzeit ausgewählten Ordner zurück
- `imapObj.search(['ON 05-Jul-2015'])` Gibt alle Nachrichten zurück, die am 5. Juli 2015 gesendet wurden
- `imapObj.search(['SINCE 01-Jan-2015', 'BEFORE 01-Feb-2015', 'UNSEEN'])` Gibt alle ungelesenen Nachrichten zurück, die im Januar 2015 gesendet wurden (alle Nachrichten, die *am und nach* dem 1. Januar und *bis zum, aber nicht am* 1. Februar gesendet wurden)
- `imapObj.search(['SINCE 01-Jan-2015', 'FROM alice@example.com'])` Gibt alle Nachrichten zurück, die seit Anfang 2015 von *alice@example.com* gesendet wurden
- `imapObj.search(['SINCE 01-Jan-2015', 'NOT FROM alice@example.com'])` Gibt alle Nachrichten zurück, die seit Anfang 2015 von allen möglichen Absendern außer *alice@example.com* gesendet wurden

- `imapObj.search(['OR FROM alice@example.com FROM bob@example.com'])` Gibt alle Nachrichten zurück, die jemals von *alice@example.com* oder *bob@example.com* gesendet wurden
- `imapObj.search(['FROM alice@example.com', 'FROM bob@example.com'])` Diese Suche gibt nichts zurück, da die Ergebnisse mit *allen* Suchschlüsseln übereinstimmen müssen. Da es aber immer nur eine Absenderadresse gibt, ist es unmöglich, dass eine Nachricht sowohl von *alice@example.com* als auch von *bob@example.com* stammt.

Die Methode `search()` gibt nicht die E-Mails als solche zurück, sondern Integerwerte, die eindeutige IDs (Unique IDs, UIDs) der E-Mails sind. Um auf den E-Mail-Inhalt zuzugreifen, übergeben Sie die UIDs an die Methode `fetch()`. Setzen Sie das Beispiel in der interaktiven Shell wie folgt fort:

```
>>> UIDs = imapObj.search(['SINCE 05-Jul-2015'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
```

Hier speichern wir die von `search()` zurückgegebene Liste der IDs für Nachrichten, die seit dem 5. Juli empfangen wurden, in der Variablen `UIDs`. Auf Ihrem Computer werden natürlich andere IDs angezeigt, da sie jeweils zu dem verwendeten E-Mail-Konto gehören. Wenn Sie in späteren Beispielen UIDs an andere Funktionen übergeben, müssen Sie natürlich diejenigen verwenden, die Sie auf Ihrem Computer empfangen haben, und nicht die aus dem Buch.

### Größeneinschränkungen

Wenn Ihre Suche eine große Menge an E-Mail-Nachrichten zutage fördert, löst Python möglicherweise die Ausnahme `imaplib.error: got more than 10000 bytes` aus. In einem solchen Fall müssen Sie die Verbindung zum IMAP-Server trennen und dann neu aufbauen, um es erneut zu versuchen.

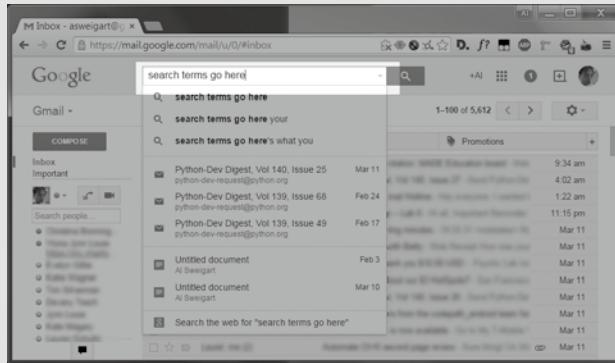
Dieser Grenzwert soll verhindern, dass Ihr Python-Programm zu viel Arbeitsspeicher verbraucht. Leider ist der Standardgrenzwert meistens zu klein. Mit dem folgenden Code können Sie ihn von 10.000 auf 10.000.000 Byte erhöhen:

```
>>> import imaplib
>>> imaplib._MAXLINE = 10000000
```

Das sollte ein weiteres Auftreten dieser Fehlermeldung verhindern. Am besten fügen Sie diese beiden Zeilen in jedes IMAP-Programm ein, das Sie schreiben.

### Die IMAPClient-Methode `gmail_search()`

Wenn Sie sich am Server `imap.gmail.com` anmelden, um auf ein Gmail-Konto zuzugreifen, können Sie eine besondere Suchfunktion des `IMAPClient`-Objekts verwenden, die das Suchfeld auf der Gmail-Webseite nachahmt (siehe Abb. 16–1).



**Abb. 16–1** Das Suchfeld auf der Webseite von Gmail

Anstatt die IMAP-Suchschlüssel zu verwenden, können Sie dadurch auf die anspruchsvollere Such-Engine von Gmail zurückgreifen. Gmail kann auch nahe verwandte Wörter finden (wenn Sie z. B. nach *driving* suchen, werden auch Vorkommen von *drive* und *drove* gefunden) und sortiert die Suchergebnisse nach ihrer Signifikanz. Sie können auch die erweiterten Suchoperatoren von Gmail nutzen (mehr dazu erfahren Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren)). Wenn Sie sich an einem Gmail-Konto anmelden, können Sie die Suchbegriffe statt an `search()` an `gmail_search()` übergeben, wie das folgende Beispiel zeigt:

```
>>> UIDs = imap0bj.gmail_search('meaning of life')
>>> UIDs
[42]
```

Ah, da ist also die E-Mail mit dem Sinn des Lebens, nach der ich gesucht habe!

### E-Mails abrufen und als gelesen markieren

Um von der Liste der UIDs zu den tatsächlichen Inhalten der E-Mails zu kommen, verwenden Sie die Methode `fetch()` des `IMAPClient`-Objekts.

Ihr erstes Argument ist die UID-Liste, das zweite gewöhnlich die Liste `['BODY[]']`, durch die die Methode angewiesen wird, den gesamten Rumpfinhalt der angegebenen E-Mails herunterzuladen.

Fahren wir mit unserem Beispiel in der interaktiven Shell fort:

```
>>> rawMessages = imapObj.fetch(UIDs, ['BODY[]'])  
>>> import pprint  
>>> pprint.pprint(rawMessages)  
{40040: {'BODY[]': 'Delivered-To: my_email_address@gmail.com\r\n'  
         'Received: by 10.76.71.167 with SMTP id '  
         '-- schnipp --  
         '\r\n'  
         '-----_Part_6000970_707736290.1404819487066--\r\n',  
         'SEQ': 5430}}
```

Wenn Sie `pprint` importieren und den in der Variablen `rawMessages` gespeicherten Rückgabewert von `fetch()` an `pprint.pprint()` übergeben, um ihn übersichtlich auszugeben, werden Sie feststellen, dass es sich bei dem Rückgabewert um ein verschachteltes Dictionary handelt, in dem die UIDs als Schlüssel fungieren. Jede Nachricht wiederum ist in einem Dictionary mit den beiden Schlüsseln `'BODY[]'` und `'SEQ'` gespeichert. Unter dem Schlüssel `'BODY[]'` befindet sich der eigentliche Rumpf der E-Mail, während der Schlüssel `'SEQ'` die *Sequenznummer* enthält, also eine laufende Nummer, die eine ähnliche Rolle spielt wie die UID. Sie können sie getrost ignorieren.

Der Nachrichteninhalt unter dem Schlüssel `'BODY[]'` ist jedoch leider ziemlich unverständlich, denn er liegt im Format RFC 822 vor, das eigens für IMAP-Server gedacht ist. Sie müssen dieses Format nicht beherrschen, denn weiter hinten in diesem Kapitel sehen wir uns das Modul `pymail` an, das daraus verständlichen Text macht.

Bei der Auswahl des zu sortierenden Ordners haben Sie `select_folder()` mit dem Schlüsselwortargument `readonly=True` aufgerufen, um zu verhindern, dass Sie versehentlich E-Mails löschen. Allerdings führt das auch dazu, dass E-Mails nicht als gelesen markiert werden, wenn Sie sie mit `fetch()` abholen. Wollen Sie jedoch eine solche Kennzeichnung durchführen, obwohl sich der ausgewählte Ordner im Schreibschutzmodus befindet, können Sie ihn mit einem weiteren Aufruf von `select_folder()` erneut auswählen und dabei `readonly=False` übergeben:

```
>>> imapObj.select_folder('INBOX', readonly=False)
```

## E-Mail-Adressen aus einer Rohnachricht gewinnen

Die von `fetch()` zurückgegebenen Rohnachrichten sind nicht sehr nützlich, wenn Sie einfach nur Ihre E-Mails lesen wollen. Dafür gibt es jedoch das Modul `pymail`, das die Rohnachrichten analysiert und als `PyZMessage`-Objekte zurückgibt. Darin

sind Betreff, Funk, An- und Von-Feld und andere Teile der E-Mail für Ihren Python-Code gut zugänglich.

Fahren Sie wie folgt mit unserem Beispiel in der interaktiven Shell fort (wobei Sie natürlich nicht die hier gezeigten UIDs verwenden, sondern diejenigen, die Sie von Ihrem eigenen E-Mail-Konto abgerufen haben):

```
>>> import pyzmail  
>>> message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])
```

Nach dem Import von `pyzmail` erstellen Sie ein `PyzMessage`-Objekt für eine E-Mail, indem Sie die Funktion `pyzmail.PyzMessage.factory()` aufrufen und ihr den `'BODY[]'`-Abschnitt der Rohnachricht übergeben. Das Ergebnis speichern Sie in `message`. Das `PyzMessage`-Objekt verfügt über mehrere Methoden, mit denen es ganz einfach ist, an die Betreffzeile, die Absender- und die Empfängeradressen zu kommen. Die Methode `get_subject()` gibt den Betreff als einfachen Stringwert zurück. Das Ergebnis von `get_addresses()` ist eine Liste der Adressen in dem als Argument übergebenen Feld. Betrachten Sie als Beispiel die folgenden Methodenaufrufe:

```
>>> message.get_subject()  
'Hello!'  
>>> message.get_addresses('from')  
[('Edward Snowden', 'esnowden@nsa.gov')]  
>>> message.get_addresses('to')  
[(Jane Doe', 'my_email_address@gmail.com')]  
>>> message.get_addresses('cc')  
[]  
>>> message.get_addresses('bcc')  
[]
```

Das Argument von `get_addresses()` ist `'from'`, `'to'`, `'cc'` oder `'bcc'`, der Rückgabewert eine Liste von Tupeln, die jeweils zwei Strings enthalten: Der erste ist der mit der E-Mail-Adresse verknüpfte Name, der zweite die Adresse selbst. Enthält das angeforderte Feld keine Adresse, gibt die Methode eine leere Liste zurück, wie es in dem vorstehenden Beispiel bei den Feldern für die Kopie (`'cc'`) und die Blindkopie (`'bcc'`) der Fall ist.

## Den Rumpf aus einer Rohnachricht gewinnen

E-Mails können als reiner Text, im HTML-Format oder in einer Mischform gesendet werden. Neben Text können HTML-E-Mails auch Farben, Schriftarten, Bilder und andere Gestaltungsmerkmale aufweisen, die sie wie eine kleine Webseite erscheinen lassen. Im `PyzMessage`-Objekt einer reinen Text-E-Mail sind alle `html_part`-Attribute auf `None` gesetzt, bei einer reinen HTML-Nachricht alle `text_part`-Attribute.

Anderenfalls verfügt der `text_part`- oder der `html_part`-Wert über die Methode `get_payload()`, die den E-Mail-Rumpf als Wert vom Datentyp `bytes` zurückgibt (den wir in diesem Buch nicht ausführlicher erklären können) – also immer noch nicht als einen lesbaren Stringwert! Argh! Als letzten Schritt müssen wir daher die Methode `decode()` für diesen Bytewert aufrufen. Sie nimmt die im Argument `text_part.charset` oder `html_part.charset` gespeicherte Zeichenkodierung als Argument entgegen und gibt den E-Mail-Rumpf endlich als String zurück.

Führen Sie das Beispiel in der interaktiven Shell wie folgt fort:

```
>>> message.text_part != None    ❶
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
'So long, and thanks for all the fish!\r\n\r\n-A1\r\n'    ❷
>>> message.html_part != None   ❸
True
>>> message.html_part.get_payload().decode(message.html_part.charset)  ❹
'<div dir="ltr"><div>So long, and thanks for all the fish!<br><br></div>-A1
<br></div>\r\n'
```

Da die E-Mail, mit der wir hier arbeiten, sowohl über Text- als auch über HTML-Inhalt verfügt, sind die Attribute `text_part` und `html_part` des in `message` gespeicherten `PyzMessage`-Objekts beide nicht `None` (❶) (❸). Durch den Aufruf von `get_payload()` für `text_part` und den anschließenden Aufruf von `decode()` für den daraus resultierenden Bytewert erhalten wir einen String der Textversion dieser E-Mail (❷). Wenn wir `get_payload()` und `decode()` für `html_part` aufrufen, ergibt sich dagegen ein String mit der HTML-Version (❹).

## E-Mails löschen

Um E-Mails zu löschen, übergeben Sie der Methode `delete_messages()` des IMAP-Client-Objekts eine Liste ihrer UIDs. Dadurch werden die E-Mails jedoch nur mit dem Flag `\Deleted` versehen. Erst ein Aufruf von `expunge()` entfernt endgültig alle als `\Deleted` gekennzeichneten E-Mails aus dem aktuellen Ordner. Betrachten Sie dazu das folgende Beispiel:

```
>>> imapObj.select_folder('INBOX', readonly=False)    ❶
>>> UIDs = imapObj.search(['ON 09-Jul-2015'])    ❷
>>> UIDs
[40066]
>>> imapObj.delete_messages(UIDs)
{40066: ('\\Seen', '\\Deleted')}    ❸
>>> imapObj.expunge()
('Success', [(5452, 'EXISTS')])
```

Hier rufen wir `select_folder()` für das `IMAPClient`-Objekt auf und übergeben '`INBOX`' als erstes Argument, um den Posteingang auszuwählen. Mit dem Schlüsselwortargument `readonly=False` sorgen wir außerdem dafür, dass wir E-Mails löschen können (❶). Wir durchsuchen den Posteingang nach E-Mails, die an einem bestimmten Datum eingegangen sind, und speichern die zurückgegebenen UIDs im Dictionary `UIDs` (❷). Wenn wir danach `delete_messages()` aufrufen und UIDs übergeben, erhalten wir ein Dictionary mit den Nachrichten-IDs als Schlüssel und einem Tupel mit den Flags der betreffenden Nachricht als Wert. Dieses Tupel enthält jetzt auf jeden Fall auch das Flag `\Deleted` (❸). Um die so gekennzeichneten Nachrichten endgültig zu löschen, rufen wir die Methode `expunge()` auf, die eine Erfolgsmeldung zurückgeben sollte. Einige E-Mail-Provider, darunter Gmail, warten jedoch nicht darauf, dass der IMAP-Client den Befehl zum endgültigen Löschen gibt, sondern entfernen die von `delete_messages()` gekennzeichneten Nachrichten automatisch.

### Die Verbindung zum IMAP-Server trennen

Um die Verbindung zum IMAP-Server nach dem Abrufen oder Löschen von Nachrichten wieder zu trennen, rufen Sie einfach die `IMAPClient`-Methode `logout()` auf:

```
>>> imapObj.logout()
```

Wenn Ihr Programm mehrere Minuten lang läuft, kann es zu einer *Zeitüberschreitung (Timeout)* kommen, wodurch die Verbindung automatisch getrennt wird. In diesem Fall löst der nächste Aufruf einer Methode für das `IMAPClient`-Objekt eine Ausnahme aus:

```
imaplib.abort: socket error: [WinError 10054] An existing connection was  
forcibly closed by the remote host
```

Das Programm muss dann erneut `imapclient.IMAPClient()` aufrufen, um die Verbindung wiederherzustellen.

Geschafft! Wir mussten eine ganze Menge Tricks und Kniffe anwenden, aber nun können Sie dafür sorgen, dass sich Ihre Python-Programme an einem E-Mail-Konto anmelden und E-Mails abrufen.

## Projekt: E-Mails über ausstehende Mitgliedsbeiträge senden

Nehmen wir an, Sie haben sich »freiwillig« gemeldet, um die ausstehenden Mitgliedsbeiträge für den Mandatory Volunteerism Club einzutreiben. Das ist eine wirklich langweilige Aufgabe: Sie müssen dazu ein Arbeitsblatt pflegen, in dem Sie eintragen, wer jeden Monat seinen Beitrag gezahlt hat, und E-Mails an diejenigen senden, die es nicht getan haben. Anstatt das Arbeitsblatt von Hand durchzugehen und immer wieder denselben Text in die E-Mails an die säumigen Zahler zu kopieren, schreiben Sie – wie sollte es auch anders sein! – ein Skript, das Ihnen diese Arbeit abnimmt.

Das Programm muss also folgende Aufgaben ausführen:

- Daten in einem Excel-Arbeitsblatt lesen
- Alle Mitglieder ermitteln, die ihren Beitrag für den letzten Monat noch nicht entrichtet haben
- Die E-Mail-Adressen dieser Mitglieder ermitteln und personalisierte Mahnungen senden

Dazu muss der Code Folgendes tun:

- Die Zellen in dem Excel-Dokument mithilfe des Moduls `openpyxl` öffnen und lesen (siehe Kapitel 12)
- Ein Dictionary der Mitglieder erstellen, die mit den Zahlungen im Rückstand sind
- Mit `smtplib.SMTP()`, `ehlo()`, `starttls()` und `login()` eine Verbindung zu einem SMTP-Server herstellen und sich daran anmelden
- Mithilfe der Methode `sendmail()` eine personalisierte E-Mail-Mahnung an alle säumigen Mitglieder senden

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das noch leere Programm als `sendDuesReminders.py`.

### Schritt 1: Die Excel-Datei öffnen

Das Excel-Arbeitsblatt, das Sie für die Mitgliedsbeiträge führen, soll den Namen `duesRecords.xlsx` tragen und so aufgebaut sein wie in Abb. 16–2. Diese Datei können Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.

	A	B	C	D	E	F	G	H
1	Member	Email	Jan 2014	Feb 2014	Mar 2014	Apr 2014	May 2014	Jun 2014
2	Alice	alice@example.com	paid	paid	paid	paid		
3	Bob	bob@example.com	paid	paid	paid	paid		
4	Carol	carol@example.com	paid	paid	paid	paid	paid	
5	David	david@example.com	paid	paid	paid	paid	paid	
6	Eve	eve@example.com	paid	paid	paid	paid	paid	
7	Fred	fred@example.com	paid	paid	paid	paid	paid	
8								
9								

**Abb. 16–2** Das Arbeitsblatt zur Aufzeichnung der monatlichen Beitragszahlungen

Dieses Arbeitsblatt enthält die Namen und E-Mail-Adressen aller Mitglieder. Für jeden Monat gibt es eine Spalte mit dem Zahlungsstand. Sobald ein Mitglied seinen Beitrag gezahlt hat, wird in die Zelle für den betreffenden Monat und das Mitglied der Text *paid* eingetragen.

Das Programm muss *duesRecords.xlsx* öffnen und mithilfe der Methode `get_highest_column()` die Spalte für den jüngsten Monat ausfindig machen. (Wie Sie mithilfe des Moduls `openpyxl` auf die Zellen in Excel-Arbeitsblättern zugreifen, können Sie in Kapitel 12 nachschlagen.) Geben Sie im Dateieditor folgenden Code ein:

```
#! python3
# sendDuesReminders.py - Sendet E-Mails auf der Grundlage des Zahlungsstands
# im Arbeitsblatt

import openpyxl, smtplib, sys

# Öffnet das Arbeitsblatt und ruft Zahlungsstand für den letzten Beitrag ab
wb = openpyxl.load_workbook('duesRecords.xlsx')      ❶
sheet = wb.get_sheet_by_name('Sheet1')                ❷

lastCol = sheet.get_highest_column()                  ❸
latestMonth = sheet.cell(row=1, column=lastCol).value ❹

# TODO: Zahlungsstand jedes einzelnen Mitglieds prüfen

# TODO: Am E-Mail-Konto anmelden

# TODO: E-Mail-Mahnungen senden
```

Nach dem Import der Module `openpyxl`, `smtplib` und `sys` öffnen wir die Datei *duesRecords.xlsx* und speichern das resultierende Workbook-Objekt in `wb` (❶).

Anschließend rufen wir *Sheet 1* ab und speichern das Worksheet-Objekt dafür in sheet (❷). Nun können wir auf die Zeilen, Spalten und Zellen des Arbeitsblatts zugreifen. Wir speichern die Nummer der letzten Spalte in lastCol (❸) und greifen dann mit Zeilennummer 1 und lastCol auf die Zelle zu, die den jüngsten Monat enthalten sollte. Den Wert dieser Zelle speichern wir in latestMonth (❹).

## Schritt 2: Alle säumigen Mitglieder finden

Im zweiten Schritt durchlaufen Sie alle Zeilen der Spalte für den jüngsten Monat außer der ersten (in der nur der Spaltenkopf steht), um zu sehen, für welche Mitglieder diese Spalte den Text *paid* enthält. Wenn ein Mitglied noch nicht gezahlt hat, entnehmen Sie seinen Namen und seine E-Mail-Adresse aus den Spalten 1 und 2 und speichern sie in unpaidMembers, dem Dictionary für säumige Zahler. Ergänzen Sie *sendDuesReminder.py* dazu um folgenden Code:

```
#! python3
# sendDuesReminders.py - Sendet E-Mails auf der Grundlage des Zahlungsstands
# im Arbeitsblatt

-- schnipp --

# Prüft den Zahlungsstand der einzelnen Mitglieder
unpaidMembers = {}
for r in range(2, sheet.get_highest_row() + 1):    ❶
    payment = sheet.cell(row=r, column=lastCol).value ❷
    if payment != 'paid':
        name = sheet.cell(row=r, column=1).value    ❸
        email = sheet.cell(row=r, column=2).value    ❹
        unpaidMembers[name] = email    ❺
```

Dieser Code erstellt das leere Dictionary unpaidMembers und durchläuft dann alle Zeilen außer der ersten (❶). Für jede Zeile wird der Wert in der Spalte für den jüngsten Monat in payment gespeichert (❷). Wenn payment nicht gleich 'paid' ist, dann wird der Wert der ersten Spalte dieser Zeile in name gespeichert (❸) und der Wert der zweiten Spalte in email (❹), die dann beide zu unpaidMembers hinzugefügt werden (❺).

## Schritt 3: Personalisierte E-Mail-Mahnungen senden

Um E-Mail-Mahnungen an alle Mitglieder zu senden, die nun auf der Liste der säumigen Zahler stehen, ergänzen Sie den Code des Programms wie folgt, wobei Sie allerdings Ihre eigene E-Mail-Adresse und die Angaben zu Ihrem Provider angeben müssen:

```

#! python3
# sendDuesReminders.py - Sendet E-Mails auf der Grundlage des Zahlungsstands
# im Arbeitsblatt

-- schnipp --

# Meldet sich am E-Mail-Konto an
smtpObj = smtplib.SMTP('smtp.gmail.com', 587)
smtpObj.ehlo()
smtpObj.starttls()
smtpObj.login('my_email_address@gmail.com', sys.argv[1])

```

Hier rufen Sie `smtplib.SMTP()` auf und übergeben Domänenname und Port des Providers, um ein SMTP-Objekt zu erstellen. Anschließend rufen Sie `ehlo()` und `starttls()` und schließlich `login()` auf, wobei Sie Ihre E-Mail-Adresse und `sys.argv[1]` übergeben, worin der String mit dem Passwort gespeichert ist. Damit Sie das Passwort nicht im Quellcode speichern müssen, ist es erforderlich, es bei jeder Ausführung des Programms an der Befehlszeile einzugeben.

Nachdem sich das Programm an dem E-Mail-Konto angemeldet hat, muss es das Dictionary `unpaidMembers` durchgehen und an alle darin enthaltenen E-Mail-Adressen eine personalisierte E-Mail schicken. Ergänzen Sie `sendDuesReminders.py` dazu wie folgt:

```

#! python3
# sendDuesReminders.py - Sendet E-Mails auf der Grundlage des Zahlungsstands
# im Arbeitsblatt

-- schnipp --

# Sendet E-Mail-Mahnungen
for name, email in unpaidMembers.items():
    body = "Subject: %s dues unpaid.\nDear %s,\nRecords show that you have
①
not paid dues for %s. Please make this payment as soon as possible.
Thank you!" %
    (latestMonth, name, latestMonth)
    print('Sending email to %s...' % email) ②
    sendmailStatus = smtpObj.sendmail('my_email_address@gmail.com',
                                      email, body) ③

    if sendmailStatus != {}: ④
        print('There was a problem sending email to %s: %s' % (email,
                                                               sendmailStatus))
smtpObj.quit()

```

Dieser Code durchläuft die Namen und E-Mail-Adressen in `unpaidMembers`. Für jedes Mitglied, das nicht gezahlt hat, stellen wir eine personalisierte E-Mail zu-

sammen, die den Namen des Mitglieds und den betreffenden Monat nennt. Diese Nachricht speichern wir in `body` (❶). Wir geben die Meldung aus, dass wir eine E-Mail an die zugehörige Adresse senden (❷), und rufen dann `sendmail()` auf, wobei wir die Adresse und den Nachrichtentext übergeben (❸). Den Rückgabewert speichern wir in `sendmailStatus`.

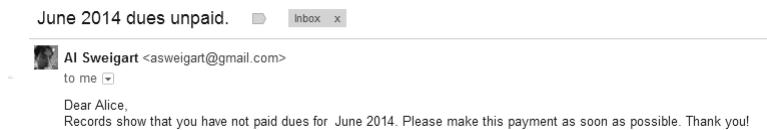
Wenn der SMTP-Server einen Fehler bei der Übermittlung einer Nachricht meldet, gibt `sendmail()` ein nicht leeres Dictionary zurück. Der letzte Teil der for-Schleife prüft, ob das zurückgegebene Dictionary leer ist (❹). Wenn nicht, werden die Empfängeradresse und das Dictionary ausgegeben.

Nachdem das Programm alle E-Mails gesendet hat, wird die Verbindung zum SMTP-Server mit `quit()` getrennt.

Die Ausgabe dieses Programms sieht wie folgt aus:

```
Sending email to alice@example.com...
Sending email to bob@example.com...
Sending email to eve@example.com...
```

Abb. 16–3 zeigt die E-Mails, die die Empfänger erhalten.



**Abb. 16–3** Eine automatisch von `sendDuesReminders.py` gesendete E-Mail

## Textnachrichten mit Twilio senden

Da man häufiger ein Mobiltelefon mit sich führt als einen Computer, bilden Textnachrichten eine unmittelbarere und zuverlässigere Methode, um Benachrichtigungen zu verschicken, als E-Mails. Da Textnachrichten kürzer sind, ist es auch eher wahrscheinlich, dass der Empfänger sie liest.

In diesem Abschnitt erfahren Sie, wie Sie sich für den kostenlosen Dienst Twilio anmelden und dessen Python-Modul nutzen können, um Textnachrichten zu senden. Bei Twilio handelt es sich um einen *SMS-Gatewaydienst*, der es Ihnen erlaubt, Textnachrichten aus Ihren Programmen heraus zu senden. Bei der kostenlosen Testversion gibt es zwar Einschränkungen, wie viele SMS Sie senden können, und außerdem wird allen Nachrichten *Sent from a Twilio trial account* vorangestellt, aber für persönliche Zwecke reicht sie meistens aus. Die kostenlose »Testversion« können Sie übrigens zeitlich unbegrenzt nutzen; es ist nicht erforderlich, nach einer bestimmten Zeit zur kostenpflichtigen Version zu wechseln.

Es gibt noch andere SMS-Gatewaydienste. Wenn Sie Twilio nicht nutzen wollen, können Sie online nach *free sms gateway*, *python sms api* oder sogar nach *twilio alternatives* suchen, um vergleichbare Dienste zu finden.

Bevor Sie ein Twilio-Konto einrichten, installieren Sie das Modul `twilio`. Einzelheiten über die Installation von Drittanbietermodulen finden Sie in Anhang A.

### Hinweis

Die Informationen in diesem Abschnitt gelten für die Nutzung in den USA. Twilio bietet SMS-Dienste auch in anderen Ländern an, wobei die Einzelheiten aber von der Beschreibung in diesem Buch abweichen können. Das Modul `twilio` und seine Funktionen funktionieren jedoch überall auf die gleiche Weise. Weitere Informationen erhalten Sie auf <http://twilio.com>.

## Ein Twilio-Konto einrichten

Füllen Sie das Registrierungsformular auf <http://twilio.com> aus. Anschließend müssen Sie die Mobiltelefonnummer angeben, zu der Sie SMS senden wollen, und bestätigen. Das soll verhindern, dass der Dienst missbraucht wird, um Spam-SMS an willkürliche Nummern zu senden.

Nachdem Sie die SMS mit der Verifizierungsnummer auf dem Telefon erhalten haben, geben Sie sie auf der Twilio-Website ein, um zu bestätigen, dass Sie der Besitzer des Mobiltelefons sind. Nun können Sie mithilfe des Moduls `twilio` Textnachrichten an diese Nummer senden.

Twilio versieht Ihr Testkonto mit einer Telefonnummer, die Sie als Absender der Textnachrichten verwenden können. Außerdem brauchen Sie noch zwei weitere Informationen, nämlich die Konto-SID und das Authentifizierungstoken. Beides finden Sie auf der Dashboard-Seite, wenn Sie sich bei Twilio angemeldet haben. Wenn Sie sich von einem Python-Programm aus anmelden, dienen diese Werte als Benutzername und Passwort für Twilio.

## Textnachrichten senden

Nachdem Sie das Modul `twilio` installiert und ein Twilio-Konto eingerichtet, Ihre Telefonnummer verifiziert, eine Twilio-Telefonnummer registriert und die Konto-SID und das Authentifizierungstoken erhalten haben, können Sie endlich von Ihren Python-Skripten aus Textnachrichten senden.

Im Vergleich zu dem Registrierungsvorgang ist der eigentliche Python-Code ziemlich einfach. Geben Sie in der interaktiven Shell Folgendes ein, wobei Sie die

Platzhalterwerte für die Variablen `accountSID`, `authToken`, `myTwilioNumber` und `myCellPhone` durch die Angaben für Ihr Konto ersetzen:

```
>>> from twilio.rest import TwilioRestClient ❶
>>> accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
>>> authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
>>> twilioCli = TwilioRestClient(accountSID, authToken) ❷
>>> myTwilioNumber = '+14955551234'
>>> myCellPhone = '+14955558888'
>>> message = twilioCli.messages.create(body='Mr. Watson - Come here - ❸
I want to see you.', from_=myTwilioNumber, to=myCellPhone)
```

Wenige Augenblicke, nachdem Sie die letzte Zeile eingegeben haben, sollten Sie die SMS mit dem Text *Sent from your Twilio trial account – Mr. Watson – Come here – I want to see you* erhalten.

Aufgrund der Art und Weise, wie das Modul `twilio` aufgebaut ist, müssen Sie zum Import `from twilio.rest import TwilioRestClient` verwenden statt einfach nur `import twilio` (❶). Speichern Sie die Konto-SID in `accountSid` und das Authentifizierungstoken in `authToken` und übergeben Sie sie an `TwilioRestClient()`. Der Aufruf dieser Funktion gibt ein `TwilioRestClient`-Objekt zurück (❷). Dessen Attribut `messages` verfügt über die Methode `create()`, mit der Sie Textnachrichten senden können. Sie ist es, die die Twilio-Server dazu bringt, Ihre SMS zu versenden. Rufen Sie `create()` auf und übergeben Sie die Schlüsselwortargumente, den Rumpf der Textnachricht sowie die Absendernummer (Ihre Twilio-Nummer) und die Empfängernummer, die Sie in `myTwilioNumber` bzw. `myCellPhone` gespeichert haben (❸).

Das von `create()` zurückgegebene `Message`-Objekt enthält Informationen über die gesendete Textnachricht, die Sie sich wie folgt ansehen können:

```
>>> message.to
'+14955558888'
>>> message.from_
'+14955551234'
>>> message.body
'Mr. Watson - Come here - I want to see you.'
```

Die Attribute `to`, `from_` und `body` enthalten die Nummer Ihres Mobiltelefons, Ihre Twilio-Nummer und die Nachricht. Beachten Sie, dass die Absendernummer im Attribut `from_` mit Unterstrich steht, nicht in `from`, da `from` ein Python-Schlüsselwort ist (Sie haben es bereits in der Form `from Modulname import *` der import-Anweisung gesehen) und nicht als Attributname verwendet werden kann. Versuchen Sie, sich in der interaktiven Shell noch weitere Attribute anzusehen:

```
>>> message.status  
'queued'  
>>> message.date_created  
datetime.datetime(2015, 7, 8, 1, 36, 18)  
>>> message.date_sent == None  
True
```

Das Attribut `status` enthält einen String, während `date_created` und `date_sent` ein `datetime`-Objekt enthalten, wenn die Nachricht erstellt bzw. gesendet wurde. Obwohl Sie die Textnachricht bereits empfangen haben, hat das Attribut `status` noch den Wert `'queued'` und `data_sent` ist auf `None` gesetzt. Das erscheint merkwürdig, liegt aber daran, dass Sie das Message-Objekt in der Variablen `message` erfasst haben, *bevor* der Text gesendet wurde. Um den aktuellen Status und das aktuelle Sendedatum zu sehen, müssen Sie das Message-Objekt erneut abrufen. Für diesen Zweck trägt jede Twilio-Nachricht eine eindeutige String-ID (SID):

```
>>> message.sid  
'SM09520de7639ba3af137c6fcb7c5f4b51'  
>>> updatedMessage = twilioCli.messages.get(message.sid) ❶  
>>> updatedMessage.status  
'delivered'  
>>> updatedMessage.date_sent  
datetime.datetime(2015, 7, 8, 1, 36, 18)
```

Wenn Sie `message.sid` eingeben, wird die lange SID der Nachricht angezeigt. Übergeben Sie sie an die `get()`-Methode des Twilio-Clients (❶), um ein neues Message-Objekt mit den aktuellen Informationen abzurufen. Darin sind `status` und `date_sent` auf die richtigen Werte gesetzt.

Das Attribut `status` kann die Werte `'queued'`, `'sending'`, `'sent'`, `'delivered'`, `'undelivered'` und `'failed'` annehmen. Diese Bezeichnungen sind im Grunde genommen selbsterklärend, aber wenn Sie mehr darüber erfahren möchten, schauen Sie in den Ressourcen auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) nach.

### Textnachrichten mit Python empfangen

Leider ist es viel komplizierter, Textnachrichten mit Twilio zu empfangen, als sie zu senden, denn dazu brauchen Sie eine Website, auf der Sie eine eigene Webanwendung ausführen. Es würde zu weit führen, das in diesem Buch zu erklären, allerdings können Sie weitere Einzelheiten in den Ressourcen auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) finden.

## Projekt: Das Modul »Just Text Me«

Die Person, der Sie von Ihren Programmen am häufigsten Textnachrichten senden werden, sind wahrscheinlich Sie selbst. SMS sind hervorragend geeignet, um sich von Ihren Programmen benachrichtigen zu lassen, während Sie selbst nicht am Computer sitzen. Wenn Sie eine langweilige Aufgabe automatisiert haben, für die das Programm mehrere Stunden braucht, dann können Sie sich mit einer Textnachricht informieren lassen, wenn der Vorgang abgeschlossen ist. Es kann auch sein, dass ein regelmäßig nach Zeitplan ausgeführtes Programm unter bestimmten Umständen Kontakt mit Ihnen aufnehmen muss, beispielsweise ein Programm, das die Wettervorhersage prüft und Ihnen mitteilt, dass Sie besser einen Regenschirm mitnehmen sollten.

Als ein einfaches Beispiel sehen wir uns das folgende kleine Python-Programm mit der Funktion `textmyself()` an, die eine als Stringargument übergebene Nachricht sendet. Geben Sie den folgenden Code im Dateieditor ein, wobei Sie als SID, Authentifizierungstoken und Telefonnummern natürlich die gültigen Werte für Ihr Konto eingeben müssen. Speichern Sie das Programm als `textMyself.py`.

```
#! python
# textMyself.py - Definiert die Funktion textmyself(), die eine als String
# übergebene Textnachricht sendet

# Voreingestellte Werte:
accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
myNumber = '+15559998888'
twilioNumber = '+15552225678'

from twilio.rest import TwilioRestClient

def textmyself(message):    ❶
    twilioCli = TwilioRestClient(accountSID, authToken)    ❷
    twilioCli.messages.create(body=message, from_=twilioNumber, to=myNumber) ❸
```

Dieses Programm speichert die Konto-SID, das Authentifizierungstoken, die Sendee und die Empfangsnummer und definiert dann die Funktion `textmyself()`, die ein einziges Argument annimmt (❶), ein `TwilioRestClient`-Objekt erstellt (❷) und `create()` mit der übergebenen Nachricht aufruft (❸).

Um die Funktion `textmyself()` auch in anderen Programmen zur Verfügung zu haben, platzieren Sie die Datei `textMyself.py` einfach in dem Ordner, in dem sich auch die ausführbare Datei von Python befindet (`C:\Python34` auf Windows, `/usr/local/lib/python3.4` auf OS X und `/usr/bin/python3` auf Linux). Nun können

andere Programme auf die Funktion zugreifen. Wenn ein Programm Ihnen eine Textnachricht senden soll, fügen Sie einfach folgende Zeilen hinzu:

```
import textmyself  
textmyself.textmyself('The boring task is finished.')
```

Die Registrierung bei Twilio und das Schreiben des Codes für das Senden von SMS sind nur einmal nötig. Danach müssen Sie immer nur zwei Zeilen Code einfügen, um sich von einem beliebigen Ihrer Programme aus über SMS benachrichtigen zu lassen.

## Zusammenfassung

Über das Internet und über die Mobilfunknetze kommunizieren wir auf viele verschiedene Weisen, wobei E-Mails und Textnachrichten jedoch vorherrschen. Auch Ihre Programme können über diese Kanäle kommunizieren, was großartige Möglichkeiten für Benachrichtigungen bietet. Sie können sogar Programme schreiben, die auf verschiedenen Computern laufen und per E-Mail miteinander kommunizieren, wobei das eine Programm E-Mails über SMTP sendet und das andere sie über IMAP empfängt.

Das Python-Modul `smtplib` enthält Funktionen zur Verwendung von SMTP, um E-Mails über den SMTP-Server Ihres Providers zu senden. Das Gegenstück dazu sind die Drittanbietermodule `imapclient` und `pymail`, mit denen Sie auf IMAP-Server zugreifen und E-Mails von dort abrufen können. IMAP ist etwas komplizierter als SMTP, dafür aber auch sehr leistungsfähig. Unter anderem ermöglicht es Ihnen, nach bestimmten E-Mails zu suchen, sie herunterzuladen und zu analysieren, um die Betreffzeile und den Rumpf als Stringwerte zu entnehmen.

Anders als bei E-Mails ist zum Senden von SMS mehr als nur eine Internetverbindung erforderlich. Zum Glück stellen Dienste wie Twilio Module bereit, mit denen Sie aus Ihren Programmen heraus Textnachrichten verschicken können. Wenn Sie die Ersteinrichtung einmal hinter sich gebracht haben, brauchen Sie zum Senden von SMS nur noch zwei Codezeilen.

Mit der Kenntnis dieser Module können Sie in Ihren Programmen genau festlegen, unter welchen Bedingungen Benachrichtigungen oder Erinnerungen gesendet werden sollen. Damit können Ihre Programme ihre Reichweite weit über den Computer hinaus ausdehnen, auf dem sie laufen!

## Wiederholungsfragen

1. Welches Protokoll dient zum Senden von E-Mails? Welches zur Überprüfung auf neu eingetroffene Mails und zum Empfangen?

2. Welche Funktionen/Methoden von `smtplib` müssen Sie aufrufen, um sich an einem SMTP-Server anzumelden?
3. Welche beiden Funktionen/Methoden von `imapclient` müssen Sie aufrufen, um sich an einem IMAP-Server anzumelden?
4. Was für ein Argument müssen Sie an `imapObj.search()` übergeben?
5. Was machen Sie, wenn Sie die Fehlermeldung `got more than 10000 bytes` erhalten?
6. Das Modul `imapclient` kümmert sich darum, die Verbindung zu einem IMAP-Server herzustellen und E-Mails zu finden. Mit welchem Modul lesen Sie die E-Mails, die `imapclient` abruft?
7. Welche drei Informationen brauchen Sie von Twilio, um Textnachrichten senden zu können?

## **Übungsprojekte**

Schreiben Sie zur Übung Programme, die folgende Aufgaben erfüllen:

### **Zufällige Zuweisung von Arbeiten**

Schreiben Sie ein Programm, das eine Liste von E-Mail-Adressen und eine Liste von zu erledigenden Arbeiten entgegennimmt, die Aufgaben zufällig an die Inhaber der E-Mail-Adressen verteilt und die Personen per E-Mail über ihre Pflichten informiert. Wenn Sie ehrgeizig sind, können Sie auch ein Verzeichnis der Aufgaben führen, die einer Person schon einmal zugewiesen wurden, damit das Programm niemandem die gleiche Arbeit zweimal hintereinander zuteilt.

Tipp: Wenn Sie der Funktion `random.choice()` eine Liste übergeben, erhalten Sie ein zufällig ausgewähltes Element dieser Liste zurück. Der betreffende Teil Ihres Codes kann wie folgt aussehen:

```
chores = ['dishes', 'bathroom', 'vacuum', 'walk dog']
randomChore = random.choice(chores)
chores.remove(randomChore) # Die Aufgabe ist vergeben und wird entfernt
```

### **Regenschirmhinweis**

In Kapitel 11 haben Sie gesehen, wie Sie mithilfe des Moduls `requests` Daten von <http://weather.gov/> abrufen. Schreiben Sie ein Programm, das läuft, kurz bevor Sie aufstehen, und prüft, ob es im Verlauf des Tages regnen wird oder nicht. Wenn ja, soll das Programm Ihnen eine SMS mit dem Hinweis schicken, dass Sie einen Regenschirm mitnehmen sollten.

## Automatischer Entregistrierer

Schreiben Sie ein Programm, das Ihr E-Mail-Konto durchforstet, alle *Unsubscribe*-Links in sämtlichen E-Mails findet und automatisch im Browser öffnet. Das Programm muss sich dazu am IMAP-Server Ihres E-Mail-Providers anmelden und alle Ihre E-Mails herunterladen. Um nach den Vorkommen des Worts *unsubscribe* in einem HTML-Linktag zu suchen, können Sie BeautifulSoup verwenden (siehe Kapitel 11).

Wenn Ihnen die Liste der URLs vorliegt, können Sie sie mit `webbrowser.open()` automatisch im Browser öffnen.

Sie müssen all diese Seiten trotzdem noch manuell durchgehen, um irgendwelche zusätzlichen Schritte auszuführen, die zum Aufheben der Registrierung erforderlich sind. In den meisten Fällen müssen Sie dazu auf einen Bestätigungslink klicken.

Allerdings nimmt Ihnen dieses Skript die Arbeit ab, all Ihre E-Mails manuell nach *Unsubscribe*-Links zu durchsuchen. Geben Sie dieses Skript auch an Ihre Freunde weiter, damit sie es für ihre eigenen E-Mail-Konten ausführen können. (Natürlich darf Ihr E-Mail-Passwort nicht im Quellcode stehen!)

## Den Computer per E-Mail steuern

Schreiben Sie ein Programm, das alle 15 Minuten ein E-Mail-Konto nach Anweisungen durchsucht, die Sie dorthin gesandt haben, und diese Anweisungen automatisch ausführt. Nehmen Sie als Beispiel ein Peer-to-Peer-Download-System wie BitTorrent. Mit kostenloser BitTorrent-Software wie qBittorrent können Sie große Mediendateien auf Ihren Computer zu Hause herunterladen. Wenn Sie dem Programm per E-Mail einen (völlig legalen) BitTorrent-Link senden, ruft es die E-Mail ab, entnimmt ihr den Link und startet dann qBittorrent, um mit dem Herunterladen der Datei zu beginnen. Dadurch kann der Computer mit der Arbeit anfangen, während Sie woanders sind, und den (völlig legalen) Download beendet haben, wenn Sie nach Hause zurückkehren.

In Kapitel 15 haben Sie erfahren, wie Sie andere Programme auf Ihrem Computer mit der Funktion `subprocess.Popen()` starten. Mit dem folgenden Aufruf können Sie das Programm qBittorrent starten und ihm eine Torrent-Datei übergeben:

```
qbProcess = subprocess.Popen(['C:\\Program Files (x86)\\qBittorrent\\
qbittorrent.exe', 'shakespeare_complete_works.torrent'])
```

Natürlich muss das Programm prüfen, ob die E-Mail tatsächlich von Ihnen stammt. Da es für Hacker ziemlich einfach ist, die Absenderadresse in einer E-Mail zu fälschen, sollten Sie in der E-Mail ein Passwort angeben. Außerdem sollte das

Programm die gefundenen E-Mails löschen, damit es nicht bei jeder Überprüfung des Posteingangs sämtliche Anweisungen wiederholt. Als zusätzliches Merkmal kann das Programm Ihnen nach jeder Ausführung eines Befehls eine Bestätigung für E-Mail oder SMS senden. Da Sie während der fraglichen Zeit nicht an dem Computer sitzen, auf dem das Programm läuft, ist es außerdem eine gute Idee, die Protokollierungsfunktionen zu verwenden (siehe Kapitel 10), um eine Protokolldatei zu führen. Sollten Fehler auftreten, können Sie darin nach Informationen suchen.

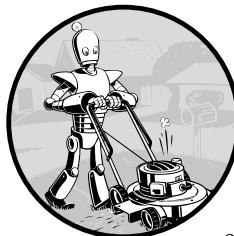
Ebenso wie viele andere BitTorrent-Anwendungen kann sich auch qBittorrent automatisch beenden, wenn der Download abgeschlossen ist. In Kapitel 15 haben Sie erfahren, wie Sie mithilfe der Methode `wait()` von `Popen`-Objekten feststellen können, ob eine gestartete Anwendung beendet wurde. Der Aufruf dieser Methode blockiert die weitere Ausführung, bis qBittorrent beendet ist. Dadurch kann das Programm feststellen, wann der Download abgeschlossen ist, und Ihnen eine entsprechende E-Mail oder SMS schicken.

Es gibt noch eine Menge Extras, die Sie diesem Projekt hinzufügen können. Wenn Sie nicht weiterkommen, können Sie eine Beispielimplementierung von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.



# 17

## Bildbearbeitung



Wenn Sie eine Digitalkamera haben oder wenn Sie auch nur Fotos von Ihrem Handy auf Facebook hochladen, haben Sie ständig mit digitalen Bilddateien zu tun. Wahrscheinlich haben Sie auch schon einmal Grafiksoftware wie Microsoft Paint oder Paintbrush oder anspruchsvollere Anwendungen wie Adobe Photoshop eingesetzt. Wenn Sie aber eine große Menge von Bildern bearbeiten müssen, kann das in Handarbeit eine ziemlich mühselige und langwierige Aufgabe sein.

Hier kommt Python ins Spiel. Das Drittanbietermodul Pillow ist für die Arbeit mit Bilddateien gemacht und bringt viele Funktionen mit, um Bilder zu beschneiden, ihre Größe zu ändern und ihren Inhalt zu bearbeiten. Damit kann Python mit Leichtigkeit Hunderte und sogar Tausende von Bildern automatisch bearbeiten – mit den gleichen gestalterischen Möglichkeiten wie Microsoft Paint, Adobe Photoshop und ähnliche Software.

## Grundlagen zur Bilddarstellung auf Computern

Um Bilder bearbeiten zu können, müssen wir wissen, wie Computer mit den Farben und Koordinaten von Bildern umgehen und wie wir selbst diese Farben und Koordinaten in Pillow beeinflussen können. Zunächst aber installieren Sie das Modul Pillow. Eine Anleitung zur Installation von Drittanbietermodulen finden Sie in Anhang A.

### Farben und RGBA-Werte

Computerprogramme stellen die Farben in einem Bild oft als *RGBA-Werte* dar. Dabei handelt es sich um Gruppen von Zahlen, die den Anteil von Rot, Grün und Blau sowie die Transparenz (Alpha) der Farbe angeben. Jeder dieser Teilwerte ist ein Integer von 0 (kein Anteil) bis 255 (Maximum). Die RGBA-Werte sind einzelnen *Pixeln* zugewiesen, wobei ein Pixel der kleinste Punkt einer einzelnen Farbe ist, den der Computermonitor anzeigen kann. (Ein Computermonitor umfasst Millionen von Pixeln.) Die RGB-Einstellung eines Pixels beschreibt genau dessen Farbe. Darüber hinaus weist ein Bild auch einen Alphawert auf, sodass sich insgesamt RGBA-Werte ergeben. Liegt hinter dem Bild ein Hintergrund, z. B. das Desktopbild des Computermonitors, legt der Alphawert fest, wie stark dieser Hintergrund durch die Pixel des Bilds »durchscheint«.

In Pillow werden RGBA-Werte durch Tupel aus vier Integerwerten dargestellt. Beispielsweise ist der Wert für die Farbe Rot (255, 0, 0, 255): Die Farbe hat einen Maximalanteil an Rot, kein Grün, kein Blau und einen maximalen Alphawert, also volle Deckkraft. Grün hat den Wert (0, 25, 0, 255), Blau (0, 0, 255, 255). Weiß, die Kombination aller Farben, ist (255, 255, 255, 255) und Schwarz, das Fehlen jeglicher Farbe, ist (0, 0, 0, 255).

Wenn eine Farbe einen Alphawert von 0 hat, ist sie unsichtbar, sodass es im Grunde nicht darauf ankommt, wie die RGB-Werte lauten. Schließlich sieht unsichtbares Rot genauso aus wie unsichtbares Schwarz.

Pillow verwendet die gleichen Standardfarbnamen wie HTML. Eine Auswahl dieser Namen und die zugehörigen Werte finden Sie in Tabelle 17–1.

Damit Sie sich die RGBA-Werte für die Farben nicht merken müssen, gibt es in Pillow die Funktion `ImageColor.getcolor()`. Sie nimmt einen String mit einem Farbnamen als erstes und den String 'RGBA' als zweites Argument entgegen und gibt ein RGBA-Tupel zurück.

Name	RGBA-Wert
White	(255, 255, 255, 255)
Green	(0, 128, 0, 255)
Gray	(128, 128, 128, 255)
Black	(0, 0, 0, 255)
Red	(255, 0, 0, 255)
Blue	(0, 0, 255, 255)
Yellow	(255, 255, 0, 255)
Purple	(128, 0, 128, 255)

**Tab. 17-1** Standardfarbnamen und zugehörige RGBA-Werte

### Die Farbräume CMYK und RGB

Schon in der Grundschule haben Sie gelernt, dass Sie aus den Farben Rot, Gelb und Blau andere Farben mischen können. Wenn Sie beispielsweise blaue und gelbe Farbe mischen, ergibt sich Grün. Dieses sogenannte *subtraktive Farbmodell* gilt für Druck- und Malfarben, Tinten, Pigmente usw. Auch Farbdrucker enthalten CMYK-Farbkatzen, um aus Cyan, Magenta, Gelb (Yellow) und Schwarz (Black) jede andere Farbe zu mischen.

Licht (wie es unter anderem auch ein Computermonitor ausstrahlt) dagegen gehorcht dem *additiven Farbmodell*. Hier kann durch die Kombination aus rotem, grünem und blauem Licht jede beliebige Farbe gebildet werden. Daher werden Farben in Computerprogrammen durch RGB-Werte dargestellt.

Probieren Sie diese Funktion wie folgt in der interaktiven Shell aus:

```
>>> from PIL import ImageColor ❶
>>> ImageColor.getcolor('red', 'RGBA') ❷
(255, 0, 0, 255)
>>> ImageColor.getcolor('RED', 'RGBA') ❸
(255, 0, 0, 255)
>>> ImageColor.getcolor('Black', 'RGBA')
(0, 0, 0, 255)
>>> ImageColor.getcolor('chocolate', 'RGBA')
(210, 105, 30, 255)
>>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
(100, 149, 237, 255)
```

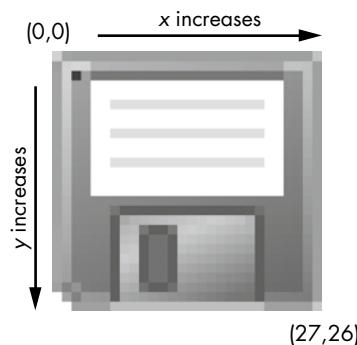
Als Erstes müssen Sie das Modul `ImageColor` von `PIL` importieren (❶) (nicht von `Pillow`; den Grund dafür werden Sie in Kürze erfahren). Da bei dem an `ImageColor.getcolor()` übergebenen Farbnamenstring nicht zwischen Groß- und Kleinschrei-

bung unterschieden wird, erhalten Sie sowohl für 'red' (❷) als auch für 'RED' (❸) dasselbe RGBA-Tupel zurück.

Sie können auch ungewöhnlichere Farbnamen übergeben, z. B. 'chocolate' oder 'Cornflower Blue'. Pillow versteht eine große Menge an Farbnamen, von 'aliceblue' bis 'whitesmoke'. Eine vollständige Liste der über 100 Standardfarbnamen finden Sie in den auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) angegebenen Quellen.

### Koordinaten und Rechtecktupel

Bildpixel werden über ihre x- und y-Koordinaten angesprochen, die ihre horizontale bzw. vertikale Lage im Bild angeben. Der *Ursprung* dieses Koordinatensystems befindet sich in der oberen linken Ecke des Bilds und hat die Koordinaten (0, 0). Die erste Zahl in einer solchen Koordinatenangabe steht für die x-Koordinate, die am Ursprung null ist und von links nach rechts zunimmt, die zweite ist die y-Koordinate, die vom Ursprung aus nach unten größer wird. Richtig, die y-Koordinate nimmt *von oben nach unten* zu, also genau andersherum als in dem Koordinatensystem, das Sie im Matheunterricht in der Schule kennengelernt haben. In Abb. 17–1 sehen Sie eine Darstellung dieses Systems.

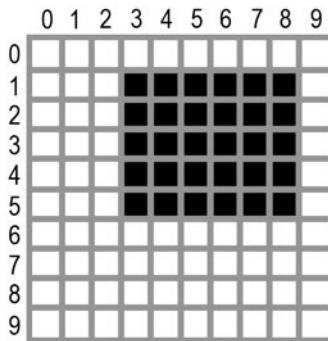


**Abb. 17–1** x- und y-Koordinaten in einem Bild von 27 x 26 Pixeln, das ein antikes Speichermedium darstellt

Viele Funktionen und Methoden von Methoden nehmen *Rechtecktupel*-Argumente entgegen. Das sind Tupel aus vier Integerkoordinaten, die für einen rechteckigen Bildbereich stehen. Bei diesen vier Integerwerten handelt es sich der Reihe nach um folgende:

- *Links*: die x-Koordinate der linken Kante des Rechtecks
- *Oben*: die y-Koordinate der Oberkante des Rechtecks
- *Rechts*: die x-Koordinate einer Linie, die ein Pixel rechts von der rechten Kante des Rechtecks liegt
- *Unten*: die y-Koordinate einer Linie, die ein Pixel unterhalb der unteren Kante des Rechtecks liegt

Das Rechteck schließt also die linke und obere Koordinate ein, nähert sich der rechten und unteren aber nur so weit wie möglich an, ohne sie zu erreichen. Beispielsweise wird das schwarze Rechteck in Abb. 17–2 durch das Rechtecktupel (3, 1, 9, 6) dargestellt.



**Abb. 17–2** Die durch das Tupel (3, 1, 9, 6) dargestellte Fläche

## Bildbearbeitung mit Pillow

Nachdem wir wissen, wie Farben und Koordinaten in Pillow angegeben werden, wollen wir dieses Modul nun zur Bildbearbeitung nutzen. Für alle Shell-Beispiele in diesem Kapitel verwenden wir das Bild aus Abb. 17–3. Sie können es von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen.

Wenn Sie das Bild *zophie.png* in Ihr aktuelles Arbeitsverzeichnis gestellt haben, können Sie es wie folgt in Python laden:

```
>>> from PIL import Image  
>>> catIm = Image.open('zophie.png')
```



**Abb. 17–3** Meine Katze Zophie. Die Kamera hat sie zehn Pfund schwerer gemacht (was für eine Katze sehr viel ist).

In diesem Code importieren Sie zunächst das Modul `Image` von Pillow und rufen dann `Image.open()` auf, wobei Sie den Dateinamen des gewünschten Bilds übergeben. Ein geladenes Bild können Sie anschließend in einer Variablen speichern, wie hier in der Variablen `CatIm`. Der Modulname von Pillow lautet `PIL`; dies dient zur Rückwärtskompatibilität mit einem älteren Modul namens Python Image Library. Aus diesem Grund müssen Sie zum Importieren `from PIL import Image` eingeben anstatt `from Pillow import Image`. Aufgrund der Art und Weise, wie die Erfinder von Pillow das Modul eingerichtet haben, können Sie nicht einfach `import PIL` schreiben, sondern Sie müssen die oben angegebene Form verwenden.

Wenn sich die Bilddatei nicht im aktuellen Arbeitsverzeichnis befindet, können Sie mithilfe der Funktion `os.chdir()` auch zu dem Ordner wechseln, in dem sie sich befindet:

```
>>> import os  
>>> os.chdir('C:\\\\folder_with_image_file')
```

Der Rückgabewert von `Image.open()` ist ein `Image`-Objekt. Das ist die Art und Weise, in der Pillow ein Bild als Python-Wert darstellt. Um ein `Image`-Objekt aus einer Bilddatei (eines beliebigen Formats) zu laden, übergeben Sie `Image.open()` einen String mit dem Dateinamen. Alle Änderungen, die Sie an dem `Image`-Objekt vornehmen, können Sie mit der Methode `save()` in einer Bilddatei (beliebigen Formats) speichern. Alle Arten der Bildbearbeitung, z. B. Drehen, Größenveränderung, Beschnitt, Zeichnen usw., nehmen Sie durch Methodenaufrufe an dem `Image`-Objekt vor.

Um die Beispiele in diesem Kapitel kürzer zu halten, setze ich voraus, dass Sie das `Image`-Modul von Pillow importiert und das Bild von Zophie in der Variablen

`catIm` gespeichert haben. Außerdem muss sich die Datei `zophie.png` im aktuellen Arbeitsverzeichnis befinden, damit `Image.open()` sie finden kann. Andernfalls müssen Sie im Stringargument für diese Funktion den vollständigen absoluten Pfad angeben.

## Arbeiten mit dem Datentyp Image

Ein `Image`-Objekt verfügt über verschiedene nützliche Attribute, die Ihnen grundlegende Informationen über die Bilddatei geben, aus der es geladen wurde: seine Breite und Höhe, seinen Dateinamen und das Grafikformat (z. B. JPEG, GIF oder PNG).

Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catIm.size
(816, 1088) ❶
>>> width, height = catIm.size ❷
>>> width ❸
816
>>> height ❹
1088
>>> catIm.filename
'zophie.png'
>>> catIm.format
'PNG'
>>> catIm.format_description
'Portable network graphics'
>>> catIm.save('zophie.jpg') ❺
```

Das Attribut `size` des in `catIm` gespeicherten `Image`-Objekts enthält ein Tupel, das die Breite und Höhe des Bilds in Pixeln angibt (❶). Diese Werte können wir den Variablen `width` und `height` zuweisen (❷), um einzeln auf die Breite (❸) und die Höhe (❹) zuzugreifen. Das Attribut `filename` gibt den Namen der ursprünglichen Datei an. Bei `format` und `format_description` handelt es sich um Strings, die das Bildformat der ursprünglichen Datei beschreiben (wobei `format_description` etwas ausführlicher ist).

Am Ende rufen wir die Methode `save()` auf und übergeben ihr `'zophie.jpg'`, um ein neues Bild mit dem Dateinamen `zophie.jpg` auf der Festplatte zu speichern (❺). Pillow erkennt die Dateierweiterung `.jpg` und speichert das Bild automatisch im JPEG-Format. Jetzt haben Sie auf Ihrer Festplatte zwei Bilder, nämlich `zophie.png` und `zophie.jpg`. Sie basieren zwar auf demselben Bild, sind aber nicht identisch, da sie ein unterschiedliches Dateiformat aufweisen.

Die Pillow-Funktion `Image.new()` gibt ebenso wie `Image.open()` ein `Image`-Objekt zurück, allerdings ist das Bild, für das dieses Objekt steht, leer. `Image.new()` nimmt folgende Argumente entgegen:

- Den String '`'RGBA'`', der den Farbmodus auf RGBA setzt. (Es gibt noch andere Modi, die wir in diesem Buch aber nicht besprechen werden.)
- Die Größe in Form eines Tupels aus zwei Integern für die Breite und Höhe des neuen Bilds.
- Die Hintergrundfarbe, die das Bild zu Anfang haben soll. Angegeben wird diese als RGBA-Wert in Form eines Tupels aus vier Integerwerten. Für dieses Argument können Sie den Rückgabewert von `ImageColor.getcolor()` verwenden, es ist aber auch möglich, einen String mit einem Standardfarbnamen zu übergeben.

Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> from PIL import Image
>>> im = Image.new('RGBA', (100, 200), 'purple') ❶
>>> im.save('purpleImage.png')
>>> im2 = Image.new('RGBA', (20, 20)) ❷
>>> im2.save('transparentImage.png')
```

Hier erstellen wir zunächst ein `Image`-Objekt für ein Bild von 100 Pixel Breite und 200 Pixeln Höhe mit violettem Hintergrund (❶). Anschließend wird das Bild in der Datei `purpleImage.png` gespeichert. Mit dem zweiten Aufruf von `Image.new()` legen wir ein weiteres `Image`-Objekt an, wobei wir diesmal (20, 20) für die Abmessungen übergeben, aber keinen Wert für die Hintergrundfarbe festlegen (❷). Wenn kein Farbargument angegeben ist, wird als Standardwert unsichtbares Schwarz verwendet, also (0, 0, 0, 0). Dieses 20 x 20 Pixel große, transparente Quadrat speichern wir anschließend in `transparentImage.png`.

## Bilder beschneiden

Beim *Beschneiden* wird ein rechteckiger Bereich innerhalb des Bilds ausgewählt und alles entfernt, was außerhalb dieses Bereichs liegt. Die Methode `crop()` für `Image`-Objekte nimmt ein Rechtecktupel entgegen und gibt ein `Image`-Objekt mit dem gewünschten Ausschnitt zurück. Der Beschnitt erfolgt nicht an dem ursprünglichen `Image`-Objekt. Es bleibt intakt, denn die Methode `crop()` gibt für den ausgeschnittenen Bereich ein neues `Image`-Objekt zurück. Denken Sie daran, dass bei Rechtecktupeln – hier also bei dem auszuschneidenden Bereich –, die linke Pixelspalte und die oberste Pixelzeile eingeschlossen sind, die rechte Spalte und die untere Zeile aber nicht.

Probieren Sie das in der interaktiven Shell aus:

```
>>> croppedIm = catIm.crop((335, 345, 565, 560))
>>> croppedIm.save('cropped.png')
```

Dabei wird ein neues `Image`-Objekt für den Ausschnitt erstellt und in `croppedIm` gespeichert. Der anschließende Aufruf von `save()` für `croppedIm` speichert den Ausschnitt als `cropped.png`. Die neue Datei `cropped.png` wird aus dem ursprünglichen Bild erstellt, wie Sie in Abb. 17–4 sehen.



**Abb. 17–4** Das neue Bild ist ein Ausschnitt des ursprünglichen Bilds.

### Bilder kopieren und in andere Bilder einfügen

Die Methode `copy()` gibt ein neues `Image`-Objekt für dasselbe Bild zurück, für dessen `Image`-Objekt sie aufgerufen wurde. Das ist praktisch, wenn Sie an einem Bild Änderungen vornehmen, aber die ursprüngliche Version aufbewahren möchten. Das können Sie wie folgt ausprobieren:

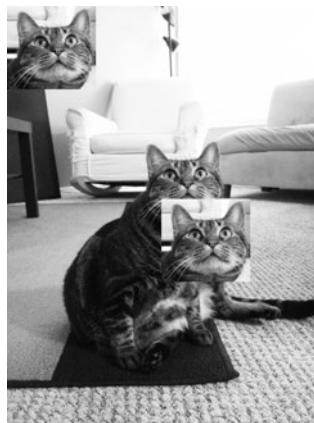
```
>>> catIm = Image.open('zophie.png')
>>> catCopyIm = catIm.copy()
```

Die Variablen `catIm` und `catCopyIm` enthalten zwei verschiedene `Image`-Objekte, die aber beide das gleiche Bild darstellen. Nun können Sie `catCopyIm` nach Belieben bearbeiten und unter einem neuen Dateinamen speichern, ohne dass sich `zophie.png` ändert. Das wollen wir ausprobieren, indem wir `catCopyIm()` mithilfe von `paste()` ändern.

Diese Methode nimmt ein Image-Objekt entgegen und fügt ein anderes Bild über diesem ersten Bild ein. Setzen Sie das Shell-Beispiel fort, indem Sie ein kleines Bild in `catCopyIm` einfügen:

```
>>> faceIm = catIm.crop((335, 345, 565, 560))
>>> faceIm.size
(230, 215)
>>> catCopyIm.paste(faceIm, (0, 0))
>>> catCopyIm.paste(faceIm, (400, 500))
>>> catCopyIm.save('pasted.png')
```

Als Erstes übergeben wir `crop()` ein Rechtecktupel für den Bereich von *zophie.png* mit Zophies Gesicht. Dadurch wird ein Image-Objekt mit einem 230 x 215 Pixel großen Ausschnitt erstellt, das wir in `faceIm` speichern. Nun können wir `faceIm` in `catCopyIm` einfügen. Die Metode `paste()` nimmt zwei Argumente entgegen: das einzufügende Image-Objekt (Quellobjekt) und ein Tupel mit den x- und y-Koordinaten, an denen die obere linke Ecke dieses Objekts im Hauptobjekt eingefügt werden soll. In dem Beispiel rufen wir `paste()` zweimal für `catCopyIm` auf, wobei wir beim ersten Mal die Koordinaten (0, 0) und beim zweiten Mal (400, 500) übergeben. Dadurch wird `faceIm` zweimal in `catCopyIm` eingefügt. Beim ersten Mal liegt die obere linke Ecke von `faceIm` im Ursprung von `catCopyIm`, beim zweiten Mal an den Koordinaten (400, 500). Schließlich speichern wir die veränderte Version von `catCopyIm` in *pasted.png*. Dieses Bild sehen Sie in Abb. 17–5.



**Abb. 17–5** Katze Zophie mit zweimal eingefügtem Gesichtsausschnitt

### Hinweis

Trotz ihrer Namen greifen die Pillow-Methoden `copy()` und `paste()` nicht auf die Zwischenablage des Computers zurück.

Bei der Verwendung der Methode `paste()` wird das Image-Objekt *direkt bearbeitet*; sie gibt also kein Image-Objekt mit dem veränderten Bild zurück. Wenn Sie `paste()` aufrufen, aber eine unveränderte Version des Originalbilds aufbewahren möchten, müssen Sie das Bild erst kopieren und `paste()` dann auf die Kopie anwenden.

Nehmen wir an, Sie möchten das gesamte Bild mit einem Kachelmuster aus Zophies Gesicht füllen wie in Abb. 17–6. Das können Sie mit zwei for-Schleifen erledigen:

```
>>> catImWidth, catImHeight = catIm.size
>>> faceImWidth, faceImHeight = faceIm.size
>>> catCopyTwo = catIm.copy()    ❶
>>> for left in range(0, catImWidth, faceImWidth):   ❷
        for top in range(0, catImHeight, faceImHeight):  ❸
            print(left, top)
            catCopyTwo.paste(faceIm, (left, top))

0 0
0 215
0 430
0 645
0 860
0 1075
230 0
230 215
-- schnipp --
690 860
690 1075
>>> catCopyTwo.save('tiled.png')
```

Hier speichern wir die Breite und Höhe von `catIm` in `catImWidth` bzw. `catImHeight`. Bei (❶) erstellen wir eine Kopie von `catIm`, die wir in `catCopyTwo` speichern. Danach beginnen wir mit den Schleifen, um `faceIm` wiederholt in `catCopyTwo` einzufügen. Die Variable `left` der äußeren for-Schleife beginnt bei 0 und wird um `faceImWidth` (230) erhöht (❷), die Variable `top` der inneren Schleife beginnt ebenfalls bei 0 und wird um `faceImHeight` (215) erhöht (❸). Diese verschachtelten for-Schleifen rufen die Werte für die Koordinaten hervor, an denen die oberen linken Ecken der `faceIm`-Bilder im Image-Objekt von `catCopyTwo` eingefügt werden sollen, sodass sich das Raster aus Abb. 17–6 ergibt. Um die Wirkungsweise der verschachtelten Schleifen beobachten zu können, geben wir den Wert von `left` und `top` aus. Nach Abschluss des Einfügevorgangs speichern wir das veränderte Bild `catCopyTwo` in `tiled.png`.



**Abb. 17–6** Verwendung von `paste()` in verschachtelten `for`-Schleifen, um das Gesicht der Katze zu duplizieren (sozusagen ein Duplikatz).

### Transparente Pixel einfügen

Normalerweise werden transparente Pixel als weiße Pixel eingefügt. Wenn das Bild, das Sie einfügen möchten, über transparente Pixel verfügt, übergeben Sie als drittes Argument das Image-Objekt, damit kein massives Viereck eingefügt wird. Dieses dritte Argument fungiert als »Maske«, also als Image-Objekt, bei dem es nur auf den Alphawert ankommt, während die Rot-, Grün- und Blauwerte ignoriert werden. Die Maske teilt der Funktion `paste()` mit, welche Pixel sie kopieren und welche sie transparent lassen soll. Die Anwendung von Masken für anspruchsvollere Zwecke kann in diesem Buch nicht erklärt werden, aber wenn Sie ein Bild einfügen wollen, in dem es transparente Pixel gibt, übergeben Sie einfach das Image-Objekt als drittes Argument.

### Die Bildgröße ändern

Wenn Sie die Methode `resize()` für ein Image-Objekt aufrufen, gibt sie ein neues Image-Objekt mit der angegebenen Breite und Höhe zurück. Als Argument nimmt sie ein Tupel aus zwei Integern entgegen, die die gewünschte Breite und Höhe angeben. Das können Sie wie folgt ausprobieren:

```
>>> width, height = catIm.size      ❶
>>> quartersizedIm = catIm.resize((int(width / 2), int(height / 2)))    ❷
>>> quartersizedIm.save('quartersized.png')
>>> svelteIm = catIm.resize((width, height + 300))      ❸
>>> svelteIm.save('svelte.png')
```

Als Erstes weisen wir hier die beiden Werte aus dem Tupel `catIm.size` den Variablen `width` und `height` zu (❶). Das macht den nachfolgenden Code übersichtlicher als die Verwendung von `catIm.size[0]` und `catIm.size[1]`.

Beim ersten Aufruf von `resize()` übergeben wir `int(width / 2)` als neue Breite und `int(height / 2)` als neue Höhe (❷). Das zurückgegebene Image-Objekt ist also nur halb so breit und halb so hoch wie das Original, d. h., es weist nur ein Viertel der ursprünglichen Größe auf. Da die Methode `resize()` in ihrem Tupelargument nur Integerwerte akzeptiert, müssen wir die beiden Divisionen in einen Aufruf von `int()` verpacken.

Bei dieser Größenänderung haben wir das Verhältnis von Breite und Höhe beibehalten. Das muss jedoch nicht sein. In der Variablen `svelteIm` speichern wir ein Image-Objekt, das genauso breit ist wie das Original, aber 300 Pixel höher (❸), sodass Zophie schlanker aussieht.

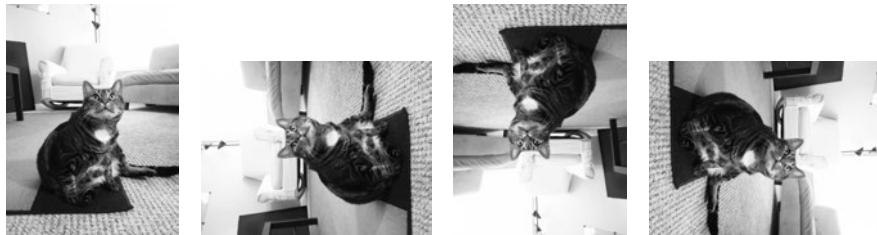
Die Methode `resize()` ändert das Image-Objekt nicht direkt, sondern gibt ein neues Image-Objekt zurück.

## Bilder drehen und spiegeln

Bilder können Sie mit der Methode `rotate()` drehen. Sie gibt ein neues Image-Objekt mit dem gedrehten Bild zurück, während das Original unberührt bleibt. Als Argument geben Sie einen einzelnen Integer- oder Fließkommawert für den Drehwinkel gegen den Uhrzeigersinn an:

```
>>> catIm.rotate(90).save('rotated90.png')
>>> catIm.rotate(180).save('rotated180.png')
>>> catIm.rotate(270).save('rotated270.png')
```

Dieses Beispiel zeigt auch, wie Sie Methoden *verketten* können: Hier wird `save()` unmittelbar für das von `rotate()` zurückgegebene Image-Objekt aufgerufen. Der erste Aufruf von `rotate()` und `save()` erzeugt ein Bild, das um 90° gegen den Uhrzeigersinn gedreht ist, und speichert es als `rotated90.png`. Beim zweiten und dritten Aufruf läuft der gleiche Vorgang ab, allerdings mit einer Drehung um 180° bzw. 270°. Die Ergebnisse sehen Sie in Abb. 17–7.



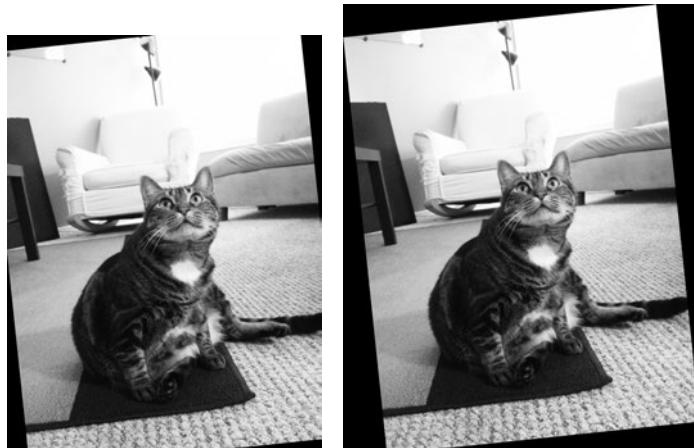
**Abb. 17–7** Das ursprüngliche Bild (links) und die um 90°, 180° und 270° gegen den Uhrzeigersinn gedrehten Bilder

Beachten Sie, dass sich Breite und Höhe bei der Drehung um 90° und 270° ändern. Bei anderen Winkeln bleiben die ursprünglichen Abmessungen des Bilds erhalten. Sollten durch die Drehung freie Flächen entstehen, werden sie unter Windows durch einen schwarzen Hintergrund gefüllt, wie Sie in Abb. 17–8 sehen. Unter OS X werden für diese freien Flächen transparente Pixel verwendet.

Die Methode `rotate()` verfügt auch über das optionale Schlüsselwortargument `expand`, das Sie auf `True` setzen können, um das neue Bild zu vergrößern, sodass das gedrehte Motiv genau hineinpasst. Betrachten Sie dazu das folgende Beispiel:

```
>>> catIm.rotate(6).save('rotated6.png')
>>> catIm.rotate(6, expand=True).save('rotated6_expanded.png')
```

Die erste Zeile dreht das Bild um 6° und speichert das Ergebnis in `rotated6.png` (links in Abb. 17–8). Beim zweiten Aufruf ist `expand` zusätzlich auf `True` gesetzt. Dieses Bild wird in `rotated6_expanded.png` gespeichert (rechts in Abb. 17–8).

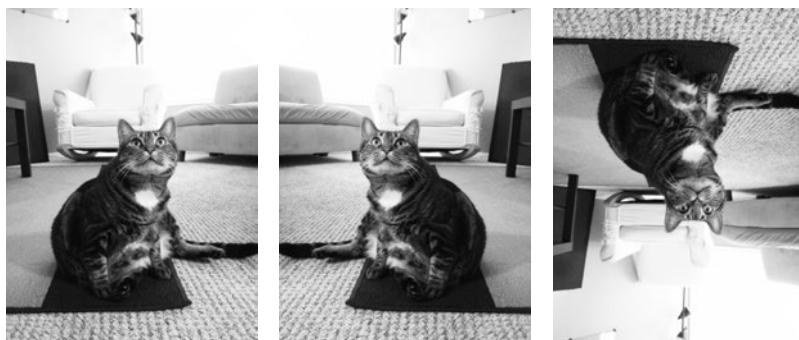


**Abb. 17–8** Das um 6° gedrehte Bild (links) und zusätzlich mit `expand=True` (rechts)

Mit der Methode `transpose()` können Sie ein Bild auch spiegeln. Dazu übergeben Sie entweder `Image.FLIP_LEFT_RIGHT` oder `Image.FLIP_TOP_BOTTOM`:

```
>>> catIm.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')
>>> catIm.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')
```

Ebenso wie `rotate()` erzeugt auch `transpose()` ein neues `Image`-Objekt. Als Erstes übergeben wir `Image.FLIP_LEFT_RIGHT`, um das Bild horizontal zu spiegeln, und speichern das Ergebnis in `horizontal_flip.png`. Zur vertikalen Spiegelung verwenden wir `Image.FLIP_TOP_BOTTOM` und legen das Bild in `vertical_flip.png` ab. Die Ergebnisse sehen Sie in Abb. 17–9.



**Abb. 17–9** Das ursprüngliche Bild (links), das horizontal gespiegelse (Mitte) und das vertikal gespiegelte (rechts)

## Einzelne Pixel ändern

Sie können auch die Farbe eines einzelnen Pixels abrufen und festlegen. Dazu verwenden Sie die Methode `getpixel()` bzw. `putpixel()`, die beide ein Tupel mit den x- und y-Koordinaten des Pixels entgegennehmen. Darüber hinaus benötigt `putpixel()` noch ein zweites Argument mit der gewünschten Farbe in Form eines RGBA- oder RGB-Tupels aus vier bzw. drei Integerwerten:

```
>>> im = Image.new('RGBA', (100, 100))    ①
>>> im.getpixel((0, 0))      ②
(0, 0, 0, 0)
>>> for x in range(100):   ③
    for y in range(50):
        im.putpixel((x, y), (210, 210, 210))  ④

>>> from PIL import ImageColor
>>> for x in range(100):   ⑤
    for y in range(50, 100):
        im.putpixel((x, y), ImageColor.getcolor('darkgray', 'RGBA'))  ⑥
```

```
>>> im.getpixel((0, 0))
(210, 210, 210, 255)
>>> im.getpixel((0, 50))
(169, 169, 169, 255)
>>> im.save('putPixel.png')
```

Bei (❶) erstellen wir ein neues Bild mit einem 100 x 100 Pixel großen, transparenten Quadrat. Wenn wir `getpixel()` für beliebige Koordinaten dieses Bilds aufrufen, erhalten wir den Rückgabewert (0, 0, 0, 0), da das Bild transparent ist (❷). Um Pixel in diesem Bild einzufärben, durchlaufen wir in verschachtelten `for`-Schleifen alle Pixel in der oberen Hälfte (❸) und rufen für sie jeweils `putpixel()` mit dem RGB-Tupel (210, 210, 210) für ein helles Grau auf (❹).

Als Nächstes wollen wir die untere Hälfte dunkelgrau färben, wir kennen die RGB-Werte dafür aber nicht. Da die Methode `putpixel()` keine Standardfarbnamen wie 'darkgray' akzeptiert, müssen wir `ImageColor.getcolor()` verwenden, um das Farbtupel für 'darkgray' zu ermitteln. Wir durchlaufen die Pixel in der unteren Hälfte des Bilds (❺) und übergeben `putpixel()` jeweils den Rückgabewert von `ImageColor.getcolor()` (❻). Ergebnis ist ein Bild mit einer hellgrauen oberen und einer dunkelgrauen unteren Hälfte, wie Sie es in Abb. 17–10 sehen. Um zu bestätigen, dass die Pixel tatsächlich die erwartete Farbe aufweisen, können Sie `getpixel()` stichprobenhaft für einige Koordinaten aufrufen. Am Ende wird das Bild als `putPixel.jpg` gespeichert.

Natürlich ist es ziemlich mühselig, ein Bild Pixel für Pixel zu zeichnen. Wenn Sie Formen zeichnen müssen, verwenden Sie die `ImageDraw`-Funktionen, die weiter hinten in diesem Kapitel erklärt werden.



**Abb. 17–10** Das fertige Bild `putPixel.jpg`

## Projekt: Ein Logo hinzufügen

Stellen Sie sich vor, Sie haben die langweilige Aufgabe, die Größe von Tausenden von Bildern zu verändern und in einer Ecke jeweils ein kleines Logo als Wasserzeichen hinzuzufügen. Mit einem einfachen Grafikprogramm wie Paintbrush oder

Paint kann das ewig dauern. Anspruchsvollere Anwendungen wie Photoshop beherrschen zwar die Stapelverarbeitung, kosten aber Hunderte von Euro. Statt dessen schreiben wir für unsere Aufgabe ein Skript.

In Abb. 17–11 sehen Sie das Logo, das in der rechten unteren Ecke jedes Bilds erscheinen soll: ein schwarzer Katzenumriss mit einem weißen Rand. Der Rest des Bilds ist transparent.



**Abb. 17–11** Das Logo, das zu dem Bild hinzugefügt werden soll

Das Programm muss also folgende Aufgaben erfüllen:

- Das Logobild laden
- Alle *.png*- und *.jpg*-Dateien im Arbeitsverzeichnis durchlaufen
- Prüfen, ob das Bild breiter oder höher als 300 Pixel ist
- Wenn ja, den größeren Wert auf 300 Pixel verringern und die andere Abmessung proportional anpassen
- Das Logobild in die Ecke einfügen
- Die geänderten Bilder in einem anderen Ordner speichern

Dazu muss der Code Folgendes tun:

- Die Datei *catlogo.png* als Image-Objekt öffnen
- Die von `os.listdir('.')` zurückgegebenen Strings durchlaufen
- Breite und Höhe des Bilds aus dem Attribut `size` abrufen
- Die neue Breite und Höhe des Bilds berechnen
- Die Größe des Bilds mit `resize()` ändern
- Das Logo mit `paste()` einfügen
- Die Änderungen mit `save()` unter dem ursprünglichen Dateinamen speichern

### Schritt 1: Das Logobild öffnen

Geben Sie den folgenden Code im Dateieditor ein und speichern Sie ihn als *resizeAndAddLogo.py*:

```
#! python3
# resizeAndAddLogo.py - Ändert die Größe aller Bilder im Arbeitsverzeichnis,
# sodass sie in ein Quadrat von 300 x 300 Pixel passen, und fügt in der
# unteren rechten Ecke catlogo.png hinzu

import os
from PIL import Image

SQUARE_FIT_SIZE = 300    ❶
LOGO_FILENAME = 'catlogo.png' ❷

logoIm = Image.open(LOGO_FILENAME) ❸
logoWidth, logoHeight = logoIm.size ❹

# TODO: Alle Dateien im Arbeitsverzeichnis durchlaufen

# TODO: Prüfen, ob die Bildgröße geändert werden muss

# TODO: Neue Breite und Höhe berechnen

# TODO: Bildgröße ändern

# TODO: Logo hinzufügen

# TODO: Änderungen speichern
```

Durch die Einrichtung der Konstanten `SQUARE_FIT_SIZE` (❶) und `LOGO_FILENAME` (❷) erleichtern wir mögliche spätere Änderungen des Programms. Wenn Sie ein anderes Symbol hinzufügen oder die maximalen Abmessungen der fertigen Bilder auf einen anderen Wert als 300 einstellen wollen, müssen Sie diese Änderungen nur an dieser einen Stelle vornehmen. (Sie können auch dafür sorgen, dass die Werte dieser Konstanten aus Befehlszeilenargumenten bezogen werden.) Ohne diese Konstanten müssten Sie den ganzen Code nach Vorkommen von 300 und 'catlogo.png' durchsuchen und durch die gewünschten neuen Werte ersetzen. Durch die Verwendung der Konstanten wird Ihr Programm flexibler.

Die Funktion `Image.open()` gibt das `Image`-Objekt für das Logo zurück (❸). Um den Code klarer zu gestalten, weisen wir die Werte von `logoIm.size` den Variablen `logoWidth` und `logoHeight` zu (❹).

Der Rest des Programms ist zunächst nur ein Skelett aus TODO-Kommentaren.

## Schritt 2: Alle Dateien durchlaufen und die Bilder öffnen

Als Nächstes müssen Sie alle `.png`- und `.jpg`-Dateien im aktuellen Arbeitsverzeichnis finden. Da das Logobild nicht zu sich selbst hinzugefügt werden soll, muss das

Programm alle Bilder überspringen, deren Dateinamen mit LOGO\_FILENAME identisch sind. Ergänzen Sie den Code wie folgt:

```
#! python3
# resizeAndAddLogo.py - Ändert die Größe aller Bilder im Arbeitsverzeichnis,
# sodass sie in ein Quadrat von 300 x 300 Pixel passen, und fügt in der
# unteren rechten Ecke catlogo.png hinzu

import os
from PIL import Image

-- schnipp --

os.makedirs('withLogo', exist_ok=True)
# Durchläuft alle Bilder im Arbeitsverzeichnis
for filename in os.listdir('.'):
    ❶ if not (filename.endswith('.png') or filename.endswith('.jpg')) \
        ❷ or filename == LOGO_FILENAME:
        continue # Überspringt Nicht-Bilddateien und die Logodatei selbst
    ❸
        im = Image.open(filename) ❹
        width, height = im.size

-- schnipp --
```

Anstatt die ursprünglichen Bilddateien zu überschreiben, legen wir die neuen, mit dem Logo versehenen Versionen im Ordner *withLogo* ab, der durch den Aufruf von `os.makedirs()` erstellt wird. Mit dem Schlüsselwortargument `exist_ok=True` verhindern wir, dass diese Funktion eine Ausnahme auslöst, falls *withLogo* bereits vorhanden ist. Während wir alle Dateien im Arbeitsverzeichnis mit `os.listdir('..')` durchlaufen (❶), prüft die lange `if`-Anweisung (❷), ob der Dateiname auf `.png` oder `.jpg` endet. Wenn nicht oder wenn es sich um das Logobild selbst handelt, überspringt die Schleife die betreffende Datei und macht mithilfe von `continue` mit der nächsten Datei weiter (❸). Dateien, deren Namen tatsächlich auf `.png` oder `.jpg` enden (und die nicht mit der Logodatei identisch sind), werden dann als `Image`-Objekt geöffnet (❹), werden und ihre Breite und Höhe in `width` und `height` gespeichert.

### Schritt 3: Die Bildgröße ändern

Das Programm ändert die Bildgröße nur, wenn die Breite oder die Höhe größer als `SQUARE_FIT_SIZE` ist (in unserem Fall also 300 Pixel). Daher stellen wir den ge-

samten Code für die Größenänderung in eine `if`-Anweisung, die die Variablen `width` und `height` prüft. Fügen Sie dem Programm folgenden Code hinzu:

```
#! python3
# resizeAndAddLogo.py - Ändert die Größe aller Bilder im Arbeitsverzeichnis,
# sodass sie in ein Quadrat von 300 x 300 Pixel passen, und fügt in der
# unteren rechten Ecke catlogo.png hinzu

import os
from PIL import Image

-- schnipp --

# Prüft, ob die Bildgröße geändert werden muss
if width > SQUARE_FIT_SIZE or height > SQUARE_FIT_SIZE:
    # Berechnet die neue Breite und Höhe
    if width > height:
        height = int((SQUARE_FIT_SIZE / width) * height) ❶
        width = SQUARE_FIT_SIZE
    else:
        width = int((SQUARE_FIT_SIZE / height) * width) ❷
        height = SQUARE_FIT_SIZE

    # Ändert die Bildgröße
    print('Resizing %s...' % (filename))
    im = im.resize((width, height)) ❸
-- schnipp --
```

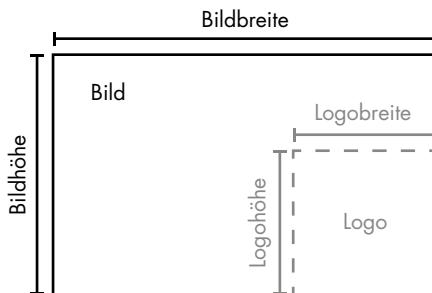
Sollte die Bildgröße geändert werden müssen, ist als Erstes zu bestimmen, ob das Bild zu breit oder zu hoch ist. Wenn `width` größer als `height` ist, muss die Höhe proportional verringert werden (❶). Der Proportionalitätsfaktor dafür ist der Wert von `SQUARE_FIT_SIZE` dividiert durch die ursprüngliche Breite. Der neue Wert für `height` ergibt sich also durch Multiplikation dieses Faktors mit dem ursprünglichen Wert von `height`. Da der Divisionsoperator einen Fließkommawert zurückgibt, `resize()` aber Integer als Abmessungen benötigt, müssen Sie das Ergebnis erst mit der Funktion `int()` in einen Integer umwandeln. Der neue Wert von `width` ist einfach `SQUARE_FIT_SIZE`.

Ist dagegen `height` größer oder gleich `width` (beide Fälle werden in der `else`-Klausel verarbeitet), werden die Rollen der Variablen `height` und `width` bei den Berechnungen einfach vertauscht (❷).

Wenn `width` und `height` die neuen Abmessungen des Bilds enthalten, werden sie an die Methode `resize()` übergeben. Das resultierende `Image`-Objekt wird in `im` gespeichert (❸).

#### Schritt 4: Logo hinzufügen und Änderungen speichern

Unabhängig davon, ob die Bildgröße geändert wurde oder nicht, muss das Logo in die untere rechte Ecke eingefügt werden. Wo genau das geschieht, hängt von der Größe des Bilds und des Logos ab. Abb. 17–12 zeigt, wie Sie die Einfügeposition berechnen. Die linke Koordinate ergibt sich aus der Bildbreite abzüglich der Logobreite, die obere Koordinate aus der Bildhöhe abzüglich der Logohöhe.



**Abb. 17–12** Die linke und die obere Koordinate für die Platzierung des Logos in der unteren rechten Ecke berechnen sich aus Bildbreite bzw. -höhe minus Logobreite/-höhe.

Nachdem der Code das Logo in das Bild eingefügt hat, soll er das veränderte Image-Objekt speichern. Ergänzen Sie das Programm wie folgt:

```
#! python3
# resizeAndAddLogo.py - Ändert die Größe aller Bilder im Arbeitsverzeichnis,
# sodass sie in ein Quadrat von 300 x 300 Pixel passen, und fügt in der
# unteren rechten Ecke catlogo.png hinzu

import os
from PIL import Image

-- schnipp --

# Prüft, ob die Bildgröße geändert werden muss

-- schnipp --

# Fügt das Logo hinzu
print('Adding logo to %s...' % (filename)) ❶
im.paste(logoIm, (width - logoWidth, height - logoHeight), logoIm) ❷

# Speichert die Änderungen
im.save(os.path.join('withLogo', filename)) ❸
```

Der neue Code gibt eine Meldung aus, die dem Benutzer mitteilt, welche Datei gerade um das Logo ergänzt wird (❶), fügt `logoIm` an den berechneten Koordinaten

in `im` ein (2) und speichert das geänderte Bild unter dem alten Dateinamen im Verzeichnis `withLogo` (3). Wenn Sie dieses Programm ausführen und sich als einzige Bilddatei `zophie.png` im Arbeitsverzeichnis befindet, erhalten Sie folgende Ausgabe:

```
Resizing zophie.png...
Adding logo to zophie.png...
```

`Zophie.png` wird in das Bild von 225 x 300 Pixel Größe geändert, das Sie in Abb. 17–13 sehen. Denken Sie daran, dass `paste()` nur dann transparente Pixel einfügt, wenn Sie `logoIm` als drittes Argument übergeben. Mit diesem Programm können Sie Hunderte von Bildern in wenigen Minuten automatisch in der Größe ändern und mit einem Logo versehen.



**Abb. 17–13** Das Bild `zophie.png` mit der neuen Größe und dem hinzugefügten Logo (links). Wenn Sie das dritte Argument weglassen, werden die transparenten Pixel des Logos als weiße Pixel eingefügt (rechts).

### Vorschläge für ähnliche Programme

Die Möglichkeit, Bilder im Stapelverarbeitungsbetrieb zu kombinieren oder in der Größe zu ändern, ist für viele Aufgaben hilfreich. Damit können Sie unter anderem Programme schreiben, die Folgendes tun:

- Text oder URLs zu Bildern hinzufügen
- Zeitstempel zu Bildern hinzufügen
- Bilder auf der Grundlage ihrer Größe in unterschiedliche Ordner kopieren oder verschieben
- Ein fast durchsichtiges Wasserzeichen zu Bildern hinzufügen, um zu verhindern, dass andere sie kopieren

## Bilder zeichnen

Wenn Sie auf einem Bild Linien, Rechtecke, Kreise oder andere einfache Formen zeichnen wollen, können Sie dazu das Pillow-Modul ImageDraw verwenden. Geben Sie dazu Folgendes in die interaktive Shell ein:

```
>>> from PIL import Image, ImageDraw  
>>> im = Image.new('RGBA', (200, 200), 'white')  
>>> draw = ImageDraw.Draw(im)
```

Nach dem Import von `Image` und `ImageDraw` erstellen wir hier ein neues, weißes Bild von 200 x 200 Pixel Größe und speichern das `Image`-Objekt in `im`. Anschließend übergeben wir das `Image`-Objekt an die Funktion `ImageDraw.Draw()`, wodurch wir ein `ImageDraw`-Objekt erhalten. Dieses Objekt verfügt über Methoden, um Formen und Text auf ein `Image`-Objekt zu zeichnen. Um das `ImageDraw`-Objekt im nachfolgenden Code einfacher nutzen zu können, speichern wir es in einer Variablen, hier `draw`.

### Formen zeichnen

Die folgenden `ImageDraw`-Methoden zeichnen verschiedene Formen auf das Bild. Die Parameter `fill` und `outline` sind optional. Werden sie nicht angegeben, wird Weiß verwendet.

#### Punkte

Die Methode `point(xy, fill)` zeichnet einzelne Pixel. Das Argument `xy` ist eine Liste der zu zeichnenden Punkte. Es kann sich dabei um eine Liste von Tupeln mit x- und y-Koordinaten wie `[(x, y), (x, y), ...]` handeln, aber auch um eine Liste von x- und y-Koordinaten nicht in Tupelform, z. B. `[x1, y1, x2, y2, ...]`. Das optionale Argument `fill` gibt die Farbe der Punkte an und ist entweder ein RGBA-Tupel oder der String eines Farbnamens wie 'red'.

#### Linien

Die Methode `line(xy, fill, width)` zeichnet eine Linie oder eine Folge von Linien. Das Argument `xy` ist eine Liste von Tupeln wie `[(x, y), (x, y), ...]` oder eine Liste von Integerwerten wie `[x1, y1, x2, y2, ...]`. Jeder der dadurch angegebenen Punkte stellt einen Verbindungspunkt der zu zeichnenden Linie dar. Das optionale Argument `fill` gibt die Farbe der Linie an und ist entweder ein RGBA-Tupel oder ein Farbname. Das ebenfalls optionale Argument `width` gibt die Linienbreite an. Der Standardwert ist 1.

## Rechtecke

Die Methode `rectangle(xy, fill, outline)` zeichnet ein Rechteck. Das Argument `xy` ist ein Rechtecktupel der Form (`left, top, right, bottom`), wobei die Werte `left` und `top` die x- und y-Koordinate der oberen linken und `right` und `bottom` die Koordinaten der unteren rechten Ecke angeben. Das optionale Argument `fill` gibt die Farbe an, mit der das Rechteck gefüllt wird. Das ebenfalls optionale Argument `outline` bestimmt die Farbe der Umrisslinie.

## Ellipsen

Die Methode `ellipse(xy, fill, outline)` zeichnet eine Ellipse. Wenn Höhe und Breite identisch sind, ergibt sich ein Kreis. Das Argument `xy` ist ein Rechtecktupel der Form (`left, top, right, bottom`). Es beschreibt das Rechteck, in das die Ellipse genau eingepasst ist. Das optionale Argument `fill` gibt die Farbe an, mit der die Ellipse gefüllt wird. Das ebenfalls optionale Argument `outline` bestimmt die Farbe der Umrisslinie.

## Polygone

Die Methode `polygon(xy, fill, outline)` zeichnet ein beliebig geformtes Polygon. Das Argument `xy` ist eine Liste von Tupeln wie `[(x, y), (x, y), ...]` oder von Integerwerten wie `[x1, y1, x2, y2, ...]`. Jeder der dadurch angegebenen Punkte stellt einen Verbindungspunkt zwischen den einzelnen Seiten des Polygons dar. Das optionale Argument `fill` gibt die Farbe an, mit der das Polygon gefüllt wird. Das ebenfalls optionale Argument `outline` bestimmt die Farbe der Umrisslinie.

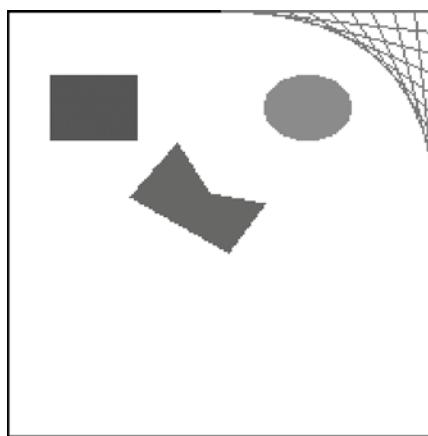
## Ein Zeichenbeispiel

Geben Sie folgenden Code in die interaktive Shell ein:

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
>>> draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)], fill='black')
❶
>>> draw.rectangle((20, 30, 60, 60), fill='blue') ❷
>>> draw.ellipse((120, 30, 160, 60), fill='red') ❸
>>> draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90), (103, 113)),
    fill='brown') ❹
>>> for i in range(100, 200, 10): ❺
        draw.line([(i, 0), (200, i - 100)], fill='green')

>>> im.save('drawing.png')
```

Nachdem wir ein Image-Objekt für ein 200 x 200 Pixel großes, weißes Bild erstellt haben, übergeben wir es an `ImageDraw.Draw()`, um ein `ImageDraw`-Objekt zu erhalten. Dieses wiederum speichern wir in `draw`, sodass wir die Zeichenmethoden für `draw` aufrufen können. Als Erstes ziehen wir einen dünnen, schwarzen Rahmen um das Bild (1). Danach zeichnen wir ein blaues Rechteck mit der oberen linken Ecke bei (20, 30) und der rechten unteren bei (60, 60) (2), eine rote Ellipse, die durch das einschließende Rechteck von (120, 30) bis (160, 60) definiert ist (3) und ein braunes, fünfeckiges Polygon (4). Mithilfe der `for`-Schleife erstellen wir außerdem ein Muster aus grünen Linien (5). Das resultierende Bild `drawing.png` sehen Sie in Abb. 17–14.



**Abb. 17–14** Das resultierende Bild `drawing.png`

Es gibt noch weitere Methoden zum Zeichnen von Formen für `ImageDraw`-Objekte. Die vollständige Dokumentation finden Sie auf <http://pillow.readthedocs.org/en/latest/reference/ImageDraw.html>.

## Text zeichnen

`ImageDraw`-Objekte verfügen auch über die Methode `text()`, um Text auf ein Bild zu zeichnen. Sie nimmt die vier Argumente `xy`, `text`, `fill` und `font` entgegen:

- `xy` ist ein Tupel aus zwei Integerzahlen, die die obere linke Ecke des Textfelds festlegen
- `text` ist der Textstring, der geschrieben werden soll
- Das optionale Argument `fill` legt die Textfarbe fest
- Das optionale Argument `font` ist ein `ImageFont`-Objekt und dient hier dazu, Schriftart und Größe des Textes festzulegen. Im nächsten Abschnitt wird es ausführlicher beschrieben.

Da es sich im Voraus oft nur schwer abschätzen lässt, welche Größe ein Textblock in einer gegebenen Schriftart einnehmen wird, bietet das Modul `ImageDraw` auch die Methode `textsize()` an. Ihr erstes Argument ist der Textstring, den Sie messen wollen, das zweite ein optionales `ImageFont`-Objekt. Die Methode gibt ein Tupel aus zwei Integern zurück, die die Breite und Höhe des Textes in der angegebenen Schriftart nennen. Damit können Sie genau berechnen, wo Sie den Text in dem Bild platzieren müssen.

Die ersten drei Argumente von `text()` erklären sich von selbst. Bevor wir die Methode aber verwenden, um Text auf ein Bild zu schreiben, sehen wir uns das optionale vierte Argument genauer an: das `ImageFont`-Objekt.

Sowohl `text()` als auch `textsize()` nehmen ein solches Objekt als ihr optionales letztes Argument entgegen. Um ein solches Objekt erstellen zu können, müssen Sie zunächst Folgendes ausführen:

```
>>> from PIL import ImageFont
```

Damit haben Sie das Pillow-Modul `ImageFont` importiert, sodass Sie nun die Funktion `ImageFont.truetype()` aufrufen können. Sie nimmt zwei Argumente entgegen. Das erste ist der String für die *TrueType-Datei* der Schriftart, die auf Ihrer Festplatte gespeichert ist. Solche Dateien haben die Endung `.ttf` und befinden sich gewöhnlich in den folgenden Ordnern:

- Auf Windows: `C:\Windows\Fonts`
- Auf OS X: `/Library/Fonts and /System/Library/Fonts`
- Auf Linux: `/usr/share/fonts/truetype`

Da Python automatisch in den entsprechenden Verzeichnissen nach Schriftarten sucht, müssen Sie diese Pfade in dem String für den Namen der TrueType-Datei nicht angeben. Wenn Python die von Ihnen angegebene Schriftart dort aber nicht finden kann, wird eine Fehlermeldung angezeigt.

Das zweite Argument von `ImageFont.truetype()` ist ein Integer für die Schriftgröße in *Punkt* (nicht Pixel!). Ein Punkt ist 1/72 Zoll. Pillow erstellt PNG-Bilder mit einer Standardauflösung von 72 Pixel pro Zoll.

Geben Sie in der interaktiven Shell folgenden Code ein und ersetzen Sie dabei `FONT_FOLDER` durch den Namen des Ordners, den Ihr Betriebssystem verwendet:

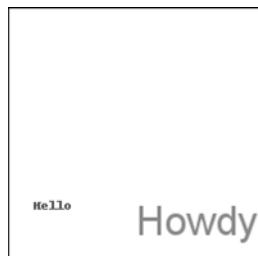
```
>>> from PIL import Image, ImageDraw, ImageFont
>>> import os
>>> im = Image.new('RGBA', (200, 200), 'white') ❶
>>> draw = ImageDraw.Draw(im) ❷
>>> draw.text((20, 150), 'Hello', fill='purple') ❸
>>> fontsFolder = 'FONT_FOLDER' # e.g. '/Library/Fonts'
```

```
>>> arialFont = ImageFont.truetype(os.path.join(fontsFolder,  
                                      'arial.ttf'), 32) ❸  
>>> draw.text((100, 150), 'Howdy', fill='gray', font=arialFont) ❹  
>>> im.save('text.png')
```

Nach dem Import von `Image`, `ImageDraw` und `os` legen wir ein `Image`-Objekt für ein neues, weißes Bild der Größe 200 x 200 an (❶) und erstellen daraus ein `ImageDraw`-Objekt (❷). Mit `text()` zeichnen wir beginnend bei (20, 150) das Wort *Hello* in Violett (❸). Da wir das optionale vierte Argument nicht übergeben haben, wird der Text in der Standardschriftart und -größe ausgegeben.

Um Schriftart und Größe festzulegen, müssen wir zunächst den Ordnernamen (z. B. `/Library/Fonts`) in `fontsFolder` speichern. Danach rufen wir `ImageFont.truetype()` auf, übergeben die gewünschte `.ttf`-Datei oder Schriftart und den Integer für die Schriftgröße (❹). Das von `ImageFont.truetype()` zurückgegebene Font-Objekt speichern wir in der Variablen `arialFont`, die wir dann als letztes Schlüsselwortargument an `text()` übergeben. Der Aufruf von `text()` bei (❹) schreibt nun *Howdy* an der Position (100, 150) in einer grauen, 32 Punkt großen Arial.

Das resultierende Bild `text.png` sehen Sie in Abb. 17–15.



**Abb. 17–15** Das resultierende Bild `text.png`

## Zusammenfassung

Bilder bestehen aus Pixeln, die jeweils über einen RGBA-Wert für die Farbe verfügen und über ihre x- und y-Koordinaten angesprochen werden können. Zwei häufig verwendete Bildformate sind JPEG und PNG. Das Modul Pillow kann mit beiden von ihnen sowie mit anderen umgehen.

Wenn ein Bild in ein `Image`-Objekt geladen wird, so werden seine Breite und seine Höhe als Tupel aus zwei Integern im Attribut `size` gespeichert. Objekte vom Datentyp `Image` haben außerdem Methoden für übliche Vorgänge der Bildbearbeitung wie `crop()`, `copy()`, `paste()`, `resize()`, `rotate()` und `transpose()`. Um das `Image`-Objekt in einer Bilddatei zu speichern, müssen Sie die Methode `save()` aufrufen.

Soll Ihr Programm Formen auf ein Bild zeichnen, verwenden Sie die Methoden von `ImageDraw` für Punkte, Linien, Rechtecke, Ellipsen und Polygone. Dieses Modul verfügt auch über Methoden, um Text in einer Schriftart und Größe Ihrer Wahl zu zeichnen.

Anspruchsvolle (und teure) Anwendungen wie Photoshop bieten eine automatische Stapelverarbeitung, aber mit Python-Skripts können Sie viele solcher Staculaufgaben kostenlos erledigen. In den vorherigen Kapiteln haben Sie Python-Programme geschrieben, die mit Textdateien, Arbeitsblättern, PDFs und anderen Formaten umgehen können. Mit dem Modul Pillow können Sie nun auch Bilder verarbeiten.

## Wiederholungsfragen

1. Was ist ein RGBA-Wert?
2. Wie bestimmen Sie mithilfe von Pillow den RGBA-Wert von Kornblumenblau ('CornflowerBlue')?
3. Was ist ein Rechtecktupel?
4. Welche Funktion gibt das `Image`-Objekt für eine Bilddatei zurück, z. B. für `zophie.png`?
5. Wie können Sie die Breite und Höhe des Bilds bestimmen, für das ein `Image`-Objekt steht?
6. Welche Methode rufen Sie auf, um ein `Image`-Objekt für ein Bild von 100 x 100 Pixel zu erstellen, bei dem das untere linke Viertel ausgespart ist?
7. Wie speichern Sie ein `Image`-Objekt als Bilddatei, nachdem Sie Änderungen daran vorgenommen haben?
8. Welches Modul enthält den Code von Pillow zum Zeichnen von Formen?
9. `Image`-Objekte haben keine Zeichenmethoden. Welche Arten von Objekten verfügen darüber? Wie können Sie an ein solches Objekt gelangen?

## Übungsprojekte

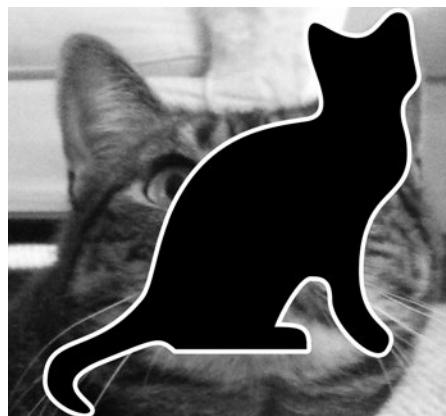
Schreiben Sie zur Übung Projekte, die folgende Aufgaben erfüllen:

### Das Logoprogramm erweitern und verbessern

Das Programm `resizeAndAddLogo.py` aus diesem Kapitel verarbeitet nur PNG- und JPEG-Bilder, obwohl Pillow noch viele andere Formate versteht. Erweitern Sie das Programm, sodass es auch GIF- und BMP-Bilder verarbeitet.

Ein anderes kleines Problem besteht darin, dass das Programm PNG- und JPEG-Dateien nur dann bearbeitet, wenn die Dateierweiterung kleingeschrieben ist. Beispielsweise wird *zophie.png* verarbeitet, *zophie.PNG* dagegen nicht. Ändern Sie den Code so, dass bei der Überprüfung der Dateierweiterung nicht zwischen Groß- und Kleinschreibung unterschieden wird.

Das Logo, das in der rechten unteren Ecke hinzugefügt wird, soll nur ein kleines Kennzeichen sein, aber wenn das Bild nicht viel größer ist als das Logo, sieht das Ergebnis wie in Abb. 17–16 aus. Ändern Sie *resizeAndAddLogo.py* so ab, dass das Bild mindestens die doppelte Breite und Höhe des Logos haben muss, damit das Logo eingefügt wird. Bei kleineren Bildern soll der Eingefügevorgang ausgelassen werden.



**Abb. 17–16** Wenn das Bild nicht viel größer ist als das Logo, ist das Ergebnis ziemlich hässlich.

### Fotoordner auf der Festplatte finden

Ich habe die schlechte Angewohnheit, Dateien von meiner Digitalkamera in temporäre Ordner irgendwo auf meiner Festplatte zu übertragen und sie dann zu vergessen. Es wäre schön, ein Programm zu haben, das die gesamte Festplatte absucht und diese übrig gebliebenen »Fotoordner« findet.

Schreiben Sie ein Programm, das alle Ordner auf Ihrer Festplatte durchsucht und mögliche Fotoordner findet. Natürlich müssen Sie dazu definieren, was ein »Fotoordner« sein soll. Eine Möglichkeit besteht darin, diese Bezeichnung für Ordner zu reservieren, bei denen mindestens die Hälfte der Dateien Fotos sind. Wie aber bestimmen wir, bei welchen Dateien es sich um Fotos handelt? Erstens muss eine Fotodatei die Erweiterung *.png* oder *.jpg* tragen. Außerdem sind Fotos große Dateien, sodass sowohl Breite als auch Höhe mehr als 500 Pixel betragen

müssen. Da die meisten Digitalkameras Bilder mit einer Breite und Höhe von mehreren tausend Pixeln erzeugen, sind wir mit diesem Wert auf der sicheren Seite.

Als kleinen Tipp sehen Sie im Folgenden das Grundgerüst des Programms:

```
#! python3
# Importieren Sie hier die Module und geben Sie in Kommentaren eine
# Beschreibung des Programms an

for foldername, subfolders, filenames in os.walk('C:\\\\'):
    numPhotoFiles = 0
    numNonPhotoFiles = 0
    for filename in filenames:
        # Prüft, ob die Dateierweiterung nicht .png oder .jpg ist
        if TODO:
            numNonPhotoFiles += 1
            continue # Fährt mit dem nächsten Dateinamen fort

        # Öffnet die Bilddatei mithilfe von Pillow.

        # Prüft, ob Breite und Höhe größer als 500 sind
        if TODO:
            # Das Bild ist groß genug, um ein Foto zu sein
            numPhotoFiles += 1
        else:
            # Das Bild ist zu klein für ein Foto
            numNonPhotoFiles += 1

    # Wenn mehr als die Hälfte der Dateien Fotos sind,
    # wird der absolute Pfad zu dem Ordner ausgegeben
    if TODO:
        print(TODO)
```

Das Programm soll den absoluten Pfad zu allen erkannten Fotoordnern auf dem Bildschirm ausgeben.

## Personalisierte Tischkarten

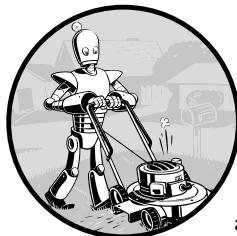
In einem Übungsprojekt in Kapitel 13 haben Sie ein Programm geschrieben, das auf der Grundlage der Gästeliste in einer Textdatei personalisierte Einladungen erstellt. Zur Ergänzung verwenden Sie nun das Modul Pillow, um Bilder für personalisierte Tischkarten zu gestalten. Erstellen Sie für jeden Gast, der in der Datei *guests.txt* unter den Ressourcen auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) aufgeführt ist, eine Bilddatei, die den Namen des Gastes und ein florales Motiv zur Dekoration erhält. Ein gemeinfreies Blumenbild finden Sie ebenfalls unter den Ressourcen auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).

Damit alle Tischkarten die gleiche Größe haben, ziehen Sie ein schwarzes Rechteck um die Kanten des Bilds. Nach dem Ausdruck können Sie diesen Rahmen dann als Markierung zum Ausschneiden nutzen. Die PNG-Dateien, die Pillow erzeugt, haben standardmäßig eine Auflösung von 72 Pixel pro Zoll. Für eine Tischkarte von 4 x 5 Zoll Größe, also ca. 10 cm x 13 cm, brauchen Sie daher ein Bild von 288 x 360 Pixel.



# 18

## Tastatur und Maus mit GUI-Automatisierung steuern



Es ist sehr praktisch, die verschiedenen Python-Module zur Bearbeitung von Arbeitsblättern, zum Herunterladen von Dateien, zum Starten von Programmen usw. zu kennen, aber es kann durchaus vorkommen, dass es für die Anwendungen, mit denen Sie arbeiten möchten, gar keine Module gibt. Die wichtigsten Werkzeuge zur Automatisierung von Aufgaben auf Ihrem Computer sind selbst geschriebene Programme, mit denen Sie die Tastatur und die Maus steuern können. Damit können Sie andere Anwendungen bedienen, indem Sie virtuelle Tastenbetätigungen und Mausklicks senden, was genauso wirkt, wie wenn Sie selbst am Computer säßen und mit der Anwendung arbeiteten. Diese Technik wird als *GUI-Automatisierung* bezeichnet (Graphical User Interface). Damit können Ihre Programme alles tun, was auch ein Mensch am Computer tun kann – außer Kaffee über die Tastatur zu kippen.

Stellen Sie sich die GUI-Automatisierung wie die Programmierung eines Roboterarms vor, der für Sie auf der Tastatur schreibt und die Maus bewegt.

Das ist besonders gut für Aufgaben geeignet, die sehr viel stumpfes Herumklicken und das Ausfüllen von Formularen erfordern.

Das Modul PyAutoGUI verfügt über Funktionen zur Simulation von Mausbewegungen, Klicks und der Betätigung des Mausrads. In diesem Kapitel können wir den Funktionsumfang dieses Moduls nicht komplett besprechen, aber auf <http://pyautogui.readthedocs.org/> finden Sie die vollständige Dokumentation.

## Das Modul PyAutoGUI installieren

Das Modul PyAutoGUI kann virtuelle Tastenbetätigungen und Mausklicks an Windows, OS X und Linux senden. Je nach vorliegendem Operationssystem müssen Sie vor PyAutoGUI eventuell noch andere Module (sogenannte *Abhängigkeiten*) installieren:

- Auf Windows sind keine anderen Module erforderlich.
- Auf OS X müssen Sie `sudo pip3 install pyobjc-framework-Quartz`, `sudo pip3 install pyobjc-core` und dann `sudo pip3 install pyobjc` ausführen.
- Auf Linux müssen Sie `sudo pip3 install python3-xlib`, `sudo apt-get install scrot`, `sudo apt-get install python3-tk` und `sudo apt-get install python3-dev` ausführen.

Danach führen Sie `pip install pyautogui` aus (bzw. `pip3` auf OS X und Linux), um PyAutoGUI zu installieren.

In Anhang A finden Sie eine ausführliche Anleitung zur Installation von Drittanbietermodulen. Um zu prüfen, ob PyAutoGUI korrekt installiert wurde, führen Sie `import pyautogui` in der interaktiven Shell aus und sehen nach, ob es irgendwelche Fehlermeldungen gibt.

## Kleine Probleme beheben

Bevor Sie sich an die GUI-Automatisierung machen, sollten Sie wissen, wie Sie möglicherweise auftretende Probleme beheben. Python kann Mausbewegungen und Tastendrücke mit atemberaubender Geschwindigkeit vollziehen, was für manche Anwendungen schon zu schnell sein mag. Wenn irgendetwas schiefgeht und das Programm immer noch munter den Mauszeiger bewegt, ist es ziemlich schwer, herauszufinden, was das Programm eigentlich macht, und vor allem, wie man es wieder anhalten kann. Wie die verhexten Besen in Goethes *Zauberlehrling*, die das Becken mit Wasser nicht nur füllen, sondern überfüllen, kann das Programm außer Kontrolle geraten, obwohl es Ihre Anweisungen genau befolgt. Wenn der Mauszeiger von selbst hierhin und dorthin wandert, ist es auch schwer,

das Programm anzuhalten, da Sie dann nicht einfach auf das IDLE-Fenster klicken können, um es zu schließen. Zum Glück gibt es verschiedene Möglichkeiten, um solche Probleme zu vermeiden oder zu beheben.

### Beenden durch Abmelden

Die vielleicht einfachste Möglichkeit, um ein außer Kontrolle geratenes GUI-Automatisierungsprogramm zu stoppen, besteht darin, sich abzumelden, wodurch alle laufenden Programme beendet werden. Auf Windows und Linux verwenden Sie dazu das Tastenkürzel **[Strg]** + **[Alt]** + **[Entf]**, auf OS X **[cmd]** + **[Umschalt]** + **[Alt]** + **[Q]**. Durch den Abmeldevorgang verlieren Sie zwar alle noch nicht gespeicherten Änderungen, aber wenigstens müssen Sie nicht auf einen kompletten Neustart des Computers warten.

### Pausen und Sicherungen

Sie können Ihr Skript anweisen, nach jedem Funktionsaufruf zu warten, sodass Sie eine Gelegenheit haben, wieder die Kontrolle über die Maus und die Tastatur zu übernehmen, wenn irgendetwas schiefgegangen sein sollte. Dazu legen Sie in der Variablen `pyautogui.PAUSE` die Länge der Pause in Sekunden fest. Wenn Sie beispielsweise `pyautogui.PAUSE = 1.5` angeben, wird jeweils anderthalb Sekunden gewartet, wenn eine aufgerufene PyAutoGUI-Funktion ihre Aufgaben erledigt hat. Anweisungen, die nicht aus PyAutoGUI stammen, sind davon nicht betroffen.

Des Weiteren verfügt PyAutoGUI über eine Sicherungsfunktion. Wenn Sie den Mauszeiger in die obere linke Bildschirmecke verschieben, löst das Modul die Ausnahme `pyautogui.FailSafeException` aus. Sie können diese Ausnahme entweder in Ihrem Programm mit `try-` und `except-`Anweisungen abfangen oder das Programm abstürzen lassen. In jedem Fall ist es damit möglich, das Programm zu beenden, indem Sie den Mauszeiger rasch so weit nach links und nach oben bewegen wie möglich. Wenn Sie diese Einrichtung ausschalten wollen, setzen Sie `pyautogui.FAILSAFE` auf `False`. Geben Sie aber für unsere nachfolgenden Beispiele Folgendes in die interaktive Shell ein:

```
>>> import pyautogui  
>>> pyautogui.PAUSE = 1  
>>> pyautogui.FAILSAFE = True
```

Damit importieren Sie `pyautogui` und sorgen für eine Pause von einer Sekunde nach jedem Funktionsaufruf. Außerdem setzen Sie `pyautogui.FAILSAFE` auf `True`, um die Sicherung einzuschalten.

## Den Mauszeiger steuern

In diesem Abschnitt lernen Sie, wie Sie mithilfe von PyAutoGUI den Mauszeiger bewegen und seine Position auf dem Bildschirm verfolgen. Dazu müssen Sie zunächst aber das Koordinatensystem von PyAutoGUI kennen.

Die Mausfunktionen von PyAutoGUI nutzen das x/y-Koordinatensystem, das Sie in Abb. 18–1 sehen und das dem Koordinatensystem für Bilder aus Kapitel 17 ähnelt. Der *Ursprung*, also der Punkt, an dem sowohl x als auch y gleich null sind, befindet sich in der oberen linken Ecke des Bildschirms. Die x-Koordinate wird von links nach rechts erhöht, die y-Koordinate von oben nach unten. Alle Koordinaten sind positive Integer; es gibt keine negativen Koordinaten.

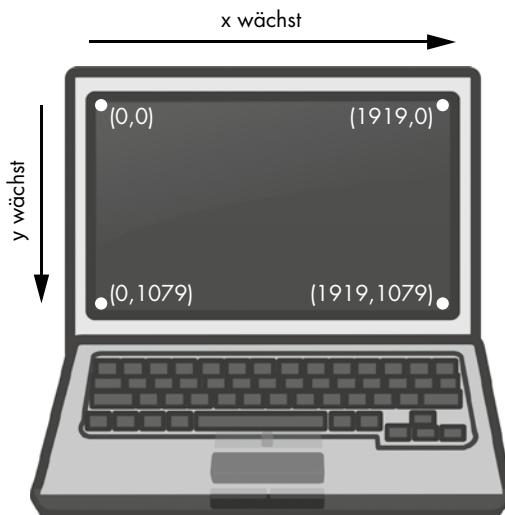


Abb. 18–1 Koordinatensystem für einen Computermonitor mit einer Auflösung von 1920 x 1080

Die *Auflösung* gibt an, wie viele Pixel in der Breite und Höhe des Bildschirms dargestellt werden. Bei einer Auflösung von 1920 x 1080 hat die untere rechte Ecke die Koordinaten (1919, 1079).

Die Funktion `pyautogui.size()` gibt ein Tupel aus zwei Integern für die Breite und Höhe des Bildschirms in Pixeln zurück. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> import pyautogui  
>>> pyautogui.size()  
(1920, 1080)  
>>> width, height = pyautogui.size()
```

Auf einem Computer mit einer Auflösung von 1920 x 1080 ergibt diese Funktion (1920, 1080); der Rückgabewert auf Ihrem Rechner hängt von der vorliegenden Auflösung ab. Um Ihre Programme übersichtlicher zu machen, können Sie die von `pyautogui.size()` zurückgegebenen Breiten- und Höhenwerte in Variablen wie `width` und `height` speichern.

## Den Mauszeiger bewegen

Da wir nun das Koordinatensystem kennen, machen wir uns daran, den Mauszeiger zu bewegen. Die Funktion `pyautogui.moveTo()` verschiebt den Mauszeiger sofort an die angegebene Position auf dem Bildschirm. Die Integerwerte für die x- und y-Koordinaten bilden dabei das erste und zweite Argument dieser Funktion. Das optionale Schlüsselwortargument `duration` ist ein Integer- oder Fließkommawert, der angibt, wie viele Sekunden die Bewegung des Mauszeigers zum Ziel in Anspruch nehmen soll. Wenn Sie dieses Argument weglassen, wird als Standardwert 0 angenommen, sodass der Mauszeiger sofort an die Position springt. (Alle `duration`-Schlüsselwertargumente für PyAutoGUI-Funktionen sind optional.) Probieren Sie Folgendes in der interaktiven Shell aus:

```
>>> import pyautogui  
>>> for i in range(10):  
    pyautogui.moveTo(100, 100, duration=0.25)  
    pyautogui.moveTo(200, 100, duration=0.25)  
    pyautogui.moveTo(200, 200, duration=0.25)  
    pyautogui.moveTo(100, 200, duration=0.25)
```

Hierdurch wird der Mauszeiger insgesamt zehnmal im Uhrzeigersinn in einem viereckigen Muster bewegt, das durch die Koordinaten festgelegt ist. Jede Bewegung nimmt eine Viertelsekunde in Anspruch, wie das Schlüsselwortargument `duration=0.25` verlangt. Ohne dieses dritte Argument würde der Mauszeiger im Nu von einem Punkt zum nächsten teleportieren.

Die Funktion `pyautogui.moveRel()` bewegt den Mauszeiger relativ zur aktuellen Position. Der folgende Beispielcode verschiebt ihn in dem gleichen Viereckmuster wie zuvor, allerdings liegt der erste Punkt dieses Vierecks an der Stelle, an der sich der Mauszeiger bei Beginn der Ausführung gerade befindet:

```
>>> import pyautogui  
>>> for i in range(10):  
    pyautogui.moveRel(100, 0, duration=0.25)  
    pyautogui.moveRel(0, 100, duration=0.25)  
    pyautogui.moveRel(-100, 0, duration=0.25)  
    pyautogui.moveRel(0, -100, duration=0.25)
```

Auch `pyautogui.moveRel()` nimmt drei Argumente entgegen: die Anzahl der Pixel, die der Mauszeiger nach rechts gehen soll, die Anzahl der Pixel, die er sich nach unten begeben soll, und optional die Dauer der Bewegung. Bei negativen Integerwerten im ersten oder zweiten Argument wird der Mauszeiger nach links bzw. oben verschoben.

### Die Position des Mauszeigers abrufen

Die aktuelle Position des Mauszeigers können Sie mit der Funktion `pyautogui.position()` abrufen, die ein Tupel mit den aktuellen x- und y-Koordinaten beim Aufruf der Funktion zurückgibt. Probieren Sie das wie folgt in der interaktiven Shell aus, wobei Sie die Maus nach jedem Aufruf bewegen:

```
>>> pyautogui.position()
(311, 622)
>>> pyautogui.position()
(377, 481)
>>> pyautogui.position()
(1536, 637)
```

Die Rückgabewerte hängen natürlich davon ab, wohin Sie den Mauszeiger bewegt haben.

### Projekt: Wo ist mein Mauszeiger?

Für Skripts zur GUI-Automatisierung ist es sehr wichtig, die Position des Mauszeigers zu bestimmen. Allerdings lassen sich die genauen Koordinaten eines Pixels nicht einfach dadurch bestimmen, dass man einen scharfen Blick auf den Bildschirm wirft. Es wäre daher praktisch, ein Programm zu haben, das ständig die x- und y-Koordinaten des Mauszeigers anzeigt, während wir ihn über den Bildschirm bewegen.

Das Programm muss also Folgendes leisten:

- Die aktuellen x- und y-Koordinaten des Mauszeigers anzeigen
- Die Koordinaten aktualisieren, während der Mauszeiger über den Bildschirm bewegt wird

Dazu muss der Code folgende Schritte ausführen:

- Die aktuellen Koordinaten mit der Funktion `position()` abrufen
- Die zuvor ausgegebenen Koordinaten durch Ausgabe des Rückschrittzeichens `\b` auf dem Bildschirm löschen
- Die Ausnahme `KeyboardInterrupt` verarbeiten, damit der Benutzer das Programm mit `Strg + C` beenden kann

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das Programm *mouseNow.py*.

### Schritt 1: Das Modul importieren

Beginnen Sie das Programm wie folgt:

```
#! python3
# mouseNow.py - Zeigt die aktuelle Position des Mauszeigers an
import pyautogui
print('Press Ctrl-C to quit.')
#TODO: Koordinaten des Mauszeigers abrufen und ausgeben
```

Am Anfang des Programms wird das Modul `pyautogui` importiert und der Hinweis ausgegeben, dass der Benutzer zum Beenden [Strg] + [C] drücken muss.

### Schritt 2: Den Beendigungscode und die Endlosschleife einrichten

Um die aktuellen Koordinaten von `mouse.position()` kontinuierlich auszugeben, verwenden Sie eine unendliche `while`-Schleife. Der Code zur Beendigung des Programms muss die Ausnahme `KeyboardInterrupt` abfangen, die ausgelöst wird, wenn der Benutzer [Strg] + [C] drückt. Wenn Sie diese Ausnahme nicht abfangen, wird eine hässliche Traceback- und Fehlermeldung angezeigt. Ergänzen Sie das bisherige Programm wie folgt:

```
#! python3
# mouseNow.py - Zeigt die aktuelle Position des Mauszeigers an
import pyautogui
print('Press Ctrl-C to quit.')
try:
    while True:
        # TODO: Koordinaten des Mauszeigers abrufen und ausgeben
except KeyboardInterrupt: ①
    print('\nDone.') ②
```

Um die Ausnahme zu behandeln, schließen Sie die `while`-Schleife in eine `try`-Anweisung ein. Wenn der Benutzer [Strg] + [C] drückt, geht die Programm-ausführung zur `except`-Klausel über (①) und gibt *Done* in einer neuen Zeile aus (②).

### Schritt 3: Die Koordinaten des Mauszeigers abrufen und anzeigen

Der Code in der while-Schleife muss die aktuellen Mauszeigerkoordinaten abrufen, auf übersichtliche Weise formatieren und ausgeben. Fügen Sie dazu folgenden Code in die Schleife ein:

```
#! python3
# mouseNow.py - Zeigt die aktuelle Position des Mauszeigers an
import pyautogui
print('Press Ctrl-C to quit.')
-- schnipp --
    # Ruft die Koordinaten des Mauszeigers ab und gibt sie aus
    x, y = pyautogui.position()
    positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
-- schnipp --
```

Mithilfe der Mehrfachzuweisung erhalten die Variablen `x` und `y` die Werte der beiden Integer in dem von `pyautogui.position()` zurückgegebenen Tupel. Anschließend werden `x` und `y` an `str()` übergeben, um die Stringformen dieser Integerkoordinaten zu bekommen. Nun können sie mit der Stringmethode `rjust()` rechtsbündig ausgerichtet werden, sodass sie unabhängig davon denselben Platz einnehmen, ob die Koordinaten nun ein, zwei, drei oder vier Stellen aufweisen. Durch die Verkettung der ausgerichteten Koordinatenstrings mit den Beschriftungen '`X:`' und '`Y:`' erhalten Sie einen übersichtlichen String, der dann in `positionStr` gespeichert wird.

Fügen Sie am Ende des Programms noch folgenden Code hinzu:

```
#! python3
# mouseNow.py - Zeigt die aktuelle Position des Mauszeigers an
-- schnipp --
    int(positionStr, end='')
    print('\b' * len(positionStr), end='', flush=True) ❶
```

Dadurch wird `positionStr` auf dem Bildschirm ausgegeben. Das Schlüsselwortargument `end=''` von `print()` verhindert, dass am Ende der Zeile das standardmäßige Zeilenumbruchzeichen erscheint. Es ist möglich, bereits ausgegebenen Text wieder zu löschen, allerdings nur in der letzten Textzeile. Sobald ein Zeilenumbruch ausgegeben wurde, kann der davor stehende Text nicht mehr entfernt werden.

Zum Löschen des vorherigen Textes geben Sie die Maskierungssequenz `\b` für das Rückschrittzeichen aus. Dadurch wird das Zeichen am Ende der aktuellen Bildschirmzeile gelöscht. Bei ❶ wird mithilfe der Stringreplikation ein String aus lauter `\b` erzeugt, der genauso lang ist wie der in `positionStr` gespeicherte String. Dadurch wird der zuletzt ausgegebene `positionStr`-String gelöscht.

Aus technischen Gründen, deren Erklärung in diesem Buch zu weit führen würde, müssen Sie bei Aufrufen von `print()`, bei denen das Rückschrittzeichen `\b` ausgegeben wird, stets `flush=True` übergeben. Wenn Sie das weglassen, erfolgt die Aktualisierung des Bildschirms möglicherweise nicht wie erwartet.

Da die `while`-Schleife so schnell wiederholt wird, bemerken die Benutzer gewöhnlich nichts davon, dass die komplette Ausgabe gelöscht und neu ausgegeben wird. Wenn beispielsweise die x-Koordinate 563 lautet und sich der Mauszeiger um ein Pixel nach rechts bewegt, sieht es so aus, als würde nur die 3 in 563 in eine 4 geändert.

Wenn Sie das Programm ausführen, werden nur zwei Zeilen ausgegeben:

```
Press Ctrl-C to quit.  
X: 290 Y: 424
```

Die erste Zeile enthält den Hinweis, wie das Programm beendet werden kann, die zweite gibt die Koordinaten des Mauszeigers an und ändert sich, während Sie den Zeiger über den Bildschirm bewegen. Mithilfe dieses Programms können Sie die erforderlichen Koordinaten für GUI-Automatisierungsskripte genau herausfinden.

## Mausinteraktionen

Als Nächstes beschäftigen wir uns damit, wie Sie mit der Maus klicken, ziehen und scrollen.

### Klicken

Um virtuelle Mausklicks an den Computer zu senden, rufen Sie die Methode `pyautogui.click()` auf. Standardmäßig simuliert diese Methode einen Linksklick an der Stelle, an der sich der Mauszeiger gerade befindet. Soll der Klick woanders stattfinden, können Sie als optionales erstes und zweites Argument die gewünschten x- und y-Koordinaten übergeben.

Um die zu simulierende Maustaste festzulegen, geben Sie das Schlüsselwortargument `button` mit dem Wert `'left'`, `'middle'` oder `'right'` an. Beispielsweise führt `pyautogui.click(100, 150, button='left')` zu einem Linksklick auf die Koordinaten (100, 150) und `pyautogui.click(200, 250, button='right')` zu einem Rechtsklick auf (200, 250).

Geben Sie Folgendes in die interaktive Shell ein:

```
>>> import pyautogui  
>>> pyautogui.click(10, 5)
```

Dadurch bewegt sich der Mauszeiger in die obere linke Ecke des Bildschirms und führt dort einen Klick aus. Ein vollständiger Klick wird dabei als Niederdrücken und Loslassen der Maustaste ohne Bewegung des Cursors definiert. Sie können einen Klick auch ausführen, indem Sie erst die Funktion `pyautogui.mouseDown()` ausführen, die die Maustaste nur drückt, und danach `pyautogui.mouseUp()`, die sie wieder loslässt. Diese Funktionen nehmen die gleichen Argumente entgegen wie `click()`. Tatsächlich ist `click()` nur eine komfortable Kombination dieser beiden Funktionsaufrufe.

Als weitere komfortable Einrichtungen gibt es die Funktion `pyautogui.doubleClick()`, die einen Doppelklick mit der linken Maustaste ausführt, sowie `pyautogui.rightClick()` und `pyautogui.middleClick()`, die einen Klick mit der rechten bzw. mittleren Maustaste vollziehen.

## Ziehen

*Ziehen* bedeutet, die Maus zu bewegen, während eine der Maustasten gedrückt ist. Beispielsweise können Sie Dateien von einem Ordner in einen anderen verlegen, indem Sie die Dateisymbole in das gewünschte Verzeichnis ziehen, oder Termine in einer Kalenderanwendung verschieben.

Zum Ziehen an eine absolute neue Position oder an eine Position relativ zu der aktuellen verwenden Sie die Funktionen `pyautogui.dragTo()` bzw. `pyautogui.dragRel()`. Sie nehmen die gleichen Argumente entgegen wie `moveTo()` und `moveRel()`, nämlich die x-Koordinate des Ziels bzw. die horizontale Strecke der Bewegung, die y-Koordinate bzw. vertikale Strecke und optional die Dauer des Vorgangs. (Wenn sich der Mauszeiger zu schnell bewegt, erfolgt der Ziehvorgang unter OS X nicht korrekt, weshalb es empfehlenswert ist, das Schlüsselwortargument `duration` zu übergeben.)

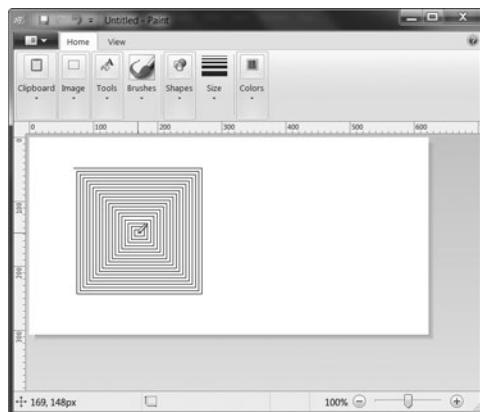
Um diese Funktionen auszuprobieren, öffnen Sie eine Zeichenanwendung wie Paint auf Windows, Paintbrush auf OS X oder GNU Paint auf Linux. (Wenn Sie kein solches Programm haben, können Sie die Onlineanwendung <http://sumopaint.com/> verwenden.)

Wählen Sie das Bleistift- oder Pinselwerkzeug aus. Geben Sie dann im Dateieditor folgenden Code ein und speichern Sie ihn als `spiralDraw.py`:

```
import pyautogui, time
time.sleep(5)      ❶
pyautogui.click() # Klick, um dem Zeichenprogramm den Fokus zu geben ❷
distance = 200
while distance > 0:
    pyautogui.dragRel(distance, 0, duration=0.2) # Nach rechts ❸
    distance = distance - 5 ❹
    pyautogui.dragRel(0, distance, duration=0.2) # Nach unten ❺
```

```
pyautogui.dragRel(-distance, 0, duration=0.2) # Nach links ❸  
distance = distance - 5  
pyautogui.dragRel(0, -distance, duration=0.2) # Nach oben
```

Nachdem Sie das Programm gestartet haben, gibt es zunächst eine Verzögerung von fünf Sekunden (❶), um Ihnen die Gelegenheit zu geben, den Mauscursor über das Fenster des Zeichenprogramms zu verschieben. Danach übernimmt *spiralDraw.py* die Kontrolle über die Maus und klickt, um dem Zeichenprogramm den *Fokus* zu geben (❷). Dadurch blinkt der Cursor in dem Fenster und alle Aktionen, die Sie vornehmen – wie Tastatureingaben oder in diesem Fall Mausbewegungen –, finden dort statt. Nachdem das Zeichenprogramm den Fokus bekommen hat, zeichnet *spiralDraw.py* ein viereckiges Spiralmuster (siehe Abb. 18–2).



**Abb. 18–2** Das Ergebnis des Beispielprogramms *spiralDraw.py*

Der Anfangswert der Variablen `distance` beträgt 200, weshalb der erste Aufruf von `dragRel()` beim ersten Durchlauf der `while`-Schleife den Cursor in 0,2 Sekunden um 200 Pixel nach rechts zieht (❸). Danach wird `distance` auf 195 verringert (❹), sodass der Mauszeiger beim zweiten Aufruf von `dragRel()` um 195 Pixel nach unten gezogen wird (❺). Der dritte Aufruf von `dragRel()` zieht den Mauszeiger horizontal um den Wert von -195, also 195 Pixel nach links (❻). Abermals wird `distance` verkleinert, diesmal auf 190, sodass der letzte Aufruf von `dragRel()` den Mauszeiger um 190 Pixel nach oben zieht. Bei jedem Schleifendurchlauf wird der Mauszeiger nun abermals nach rechts, unten, links und oben gezogen, wobei `distance` immer ein wenig kleiner ist als bei der vorherigen Iteration. Durch die Ausführung des Codes in einer Schleife wird der Mauszeiger so bewegt, dass eine Spirale entsteht.

Diese Spirale könnten Sie natürlich auch zeichnen, indem Sie die Maus mit der Hand steuern, aber dabei müssten Sie sehr genau vorgehen. PyAutoGUI erledigt diese Aufgabe in wenigen Sekunden!

### Hinweis

Der Code könnte das Bild auch mit den Zeichenfunktionen von Pillow erstellen, die Sie in Kapitel 17 kennengelernt haben. Mithilfe der GUI-Automatisierung können Sie jedoch auf die anspruchsvolleren Zeichenwerkzeuge von Grafikprogrammen zurückgreifen, z. B. Verläufe, verschiedene Pinsel und Farbfüllungen.

### Scrollen

Die letzte der Mausfunktionen von PyAutoGUI ist `scroll()`, der Sie einen Integer für die Anzahl der Einheiten übergeben, um die nach oben oder unten gescrollt werden soll. Wie groß diese Einheiten sind, hängt jeweils vom Betriebssystem und der Anwendung ab, weshalb Sie experimentieren müssen, um herauszufinden, wie weit Sie beim Scrollen kommen. Der Scrollvorgang findet an der aktuellen Position des Mauszeigers statt. Um nach oben zu scrollen, übergeben Sie eine positive Ganzzahl, um nach unten zu scrollen, eine negative. Führen Sie in der interaktiven Shell folgenden Code aus, während sich der Mauszeiger über dem IDLE-Fenster befindet:

```
>>> pyautogui.scroll(200)
```

Dabei können Sie beobachten, dass das IDLE-Fenster kurz nach oben gescrollt wird, bevor es wieder auf die alte Position zurückfällt. Letzteres geschieht, da IDLE nach dem Ausführen einer Anweisung automatisch zum unteren Rand scrollt. Versuchen Sie es stattdessen mit folgendem Code:

```
>>> import pyperclip
>>> numbers = ''
>>> for i in range(200):
    numbers = numbers + str(i) + '\n'
>>> pyperclip.copy(numbers)
```

Dadurch wird `pyperclip` importiert und der leere String `numbers` angelegt. Anschließend durchläuft der Code 200 Zahlen und hängt sie jeweils mit einem Zeilenumbruchzeichen an `numbers` an. Mit `pyperclip.copy(numbers)` werden anschließend 200 Zeilen mit Zahlen in die Zwischenablage geladen. Öffnen Sie nun

ein Dateieditorfenster und fügen Sie den Text dort ein. Dadurch steht Ihnen ein großes Textfenster zur Verfügung, in dem Sie das Scrollen ausprobieren können. Geben Sie folgenden Code in die interaktive Shell ein:

```
>>> import time, pyautogui  
>>> time.sleep(5); pyautogui.scroll(100)
```

In der zweiten Zeile geben Sie hier zwei Befehle ein, die durch ein Semikolon getrennt sind. Python führt diese Befehle so aus, als ständen sie in unterschiedlichen Zeilen. Der Unterschied besteht jedoch darin, dass die interaktive Shell Sie zwischen diesen beiden Anweisungen nicht zu Eingaben auffordert. Das ist für dieses Beispiel wichtig, da `pyautogui.scroll()` nach der Wartezeit automatisch aufgerufen werden soll. (In der interaktiven Shell kann es praktisch sein, zwei Befehle in eine Zeile zu schreiben, aber in Ihren Programmen sollten Sie jede Anweisung immer in eine eigene Zeile stellen!)

Drücken Sie die Eingabetaste, um den Code zu starten. Nun haben Sie fünf Sekunden lang Zeit, um in das Dateieditorfenster zu klicken, um ihm den Fokus zu geben. Nach dieser Verzögerung scrollt `pyautogui.scroll()` im Dateieditorfenster nach oben.

## Auf dem Bildschirm arbeiten

Ihre Programme zur GUI-Automatisierung müssen nicht blind klicken und Eingaben machen. PyAutoGUI bietet auch Möglichkeiten, um Screenshots anzufertigen. Diese Funktionen können auch ein Pillow-Objekt vom Typ `Image` zurückgeben. Wenn Sie Kapitel 17 noch nicht gelesen haben, sollten Sie das jetzt tun, bevor Sie weitermachen, und außerdem das Modul Pillow installieren.

Um die Screenshotfunktionen von PyAutoGUI zu nutzen, müssen Sie auf Linux-Computern das Programm `scrot` installieren. Führen Sie dazu in einem Terminalfenster `sudo apt-get install scrot` aus. Auf Windows und OS X brauchen Sie das nicht zu tun, sondern Sie können gleich mit dem folgenden Abschnitt weitermachen.

### Einen Screenshot aufnehmen

Um in Python einen Screenshot aufzunehmen, rufen Sie die Funktion `pyautogui.screenshot()` auf:

```
>>> import pyautogui  
>>> im = pyautogui.screenshot()
```

Die Variable `im` enthält das Image-Objekt des Screenshots, für das Sie nun die gleichen Methoden wie für jedes andere Image-Objekt aufrufen können:

```
>>> im.getpixel((0, 0))
(176, 176, 175)
>>> im.getpixel((50, 200))
(130, 135, 144)
```

Wenn Sie `getpixel()` ein Tupel mit Koordinaten wie `(0, 0)` oder `(50, 200)` übergeben, erhalten Sie die Farbe des Pixels an diesen Koordinaten zurück, und zwar in Form eines RGB-Tupels mit drei Integerwerter für den Rot-, Grün- und Blauanteil. (Da Screenshots vollständig undurchsichtig sind, gibt es keinen Alphawert.) Dadurch kann Ihr Programm »sehen«, was sich zurzeit auf dem Bildschirm befindet.

### Einen Screenshot analysieren

Nehmen wir an, Sie wollen ein GUI-Automatisierungsprogramm schreiben, in dem einer der Schritte darin besteht, auf eine graue Schaltfläche zu klicken. Bevor Sie die Methode `click()` aufrufen, können Sie einen Screenshot aufnehmen und das Pixel untersuchen, auf das das Skript zu klicken im Begriff ist. Wenn dieses Pixel nicht das gleiche Grau aufweist wie die gewünschte Schaltfläche, weiß das Programm, dass irgendetwas nicht stimmt. Es kann sein, dass das Fenster unvorhergesehen verschoben wurde oder dass ein Popup-Fenster die Schaltfläche verdeckt. Anstatt weiterzumachen – und dabei durch einen Klick auf ein falsches Element möglicherweise ein Chaos anzurichten –, kann das Programm erkennen, dass es nicht auf die richtige Stelle klickt, und sich selbst anhalten.

Die PyAutoGUI-Funktion `pixelMatchesColor()` gibt `True` zurück, wenn das Pixel an den gegebenen x- und y-Koordinaten die angegebene Farbe hat. Das erste und das zweite Element sind die Integerwerte für die Koordinaten, das dritte ist ein Tupel aus drei Integern für die RGB-Farbe, die das Pixel aufweisen soll. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
>>> im.getpixel((50, 200))    ❶
(130, 135, 144)
>>> pyautogui.pixelMatchesColor(50, 200, (130, 135, 144))   ❷
True
>>> pyautogui.pixelMatchesColor(50, 200, (255, 135, 144))   ❸
False
```

Nach der Aufnahme des Screenshots und dem Abruf des RGB-Tupels für die Pixelfarbe an den angegebenen Koordinaten (❶) werden dieselben Koordinaten und das

RGB-Tupel an `pixelMatchesColor()` übergeben (❷), was natürlich `True` ergibt. Anschließend wird `pixelMatchesColor()` erneut mit denselben Koordinaten, aber mit einem anderen RGB-Wert aufgerufen (❸). Dabei gibt die Funktion `False` zurück. Diese Methode kann immer dann sehr nützlich sein, wenn ein GUI-Automatisierungsprogramm `click()` aufrufen will. Beachten Sie, dass die Farbe an den gegebenen Koordinaten *genau* stimmen muss. Selbst bei einer kleinen Abweichung – z. B. `(255, 255, 254)` statt `(255, 255, 255)` – gibt `pixelMatchesColor()` den Wert `False` zurück.

## Projekt: Das Programm `mouseNow` erweitern

Das Programm `mouseNow.py`, das Sie weiter vorn in diesem Kapitel geschrieben haben, können Sie nun so erweitern, dass es nicht nur die x- und y-Koordinate der aktuellen Zeigerposition ausgibt, sondern auch die RGB-Farbe des Pixels unter dem Cursor. Ändern Sie den Code in der `while`-Schleife von `mouseNow.py` wie folgt:

```
#! python3
# mouseNow.py - Zeigt die aktuelle Position des Mauszeigers an
-- schnipp --
    positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
    pixelColor = pyautogui.screenshot().getpixel((x, y))
    positionStr += ' RGB: (' + str(pixelColor[0]).rjust(3)
    positionStr += ', ' + str(pixelColor[1]).rjust(3)
    positionStr += ', ' + str(pixelColor[2]).rjust(3) + ')'
    print(positionStr, end='')
-- schnipp --
```

Wenn Sie `mouseNow.py` jetzt ausführen, enthält die Ausgabe auch den RGB-Farbwert des Pixels unter dem Zeiger:

```
Press Ctrl-C to quit.
X: 406 Y: 17 RGB: (161, 50, 50)
```

Mit dieser Information und der Funktion `pixelMatchesColor()` können Sie Ihren GUI-Automatisierungsskripten nun auf ganz einfache Weise Pixelfarbprüfungen hinzufügen.

## Bilderkennung

Was tun Sie, wenn Sie im Voraus gar nicht wissen, wohin PyAutoGUI klicken soll? Dafür können Sie eine Bilderkennung einsetzen. Sie können PyAutoGUI ein Bild

des Elements übergeben, auf das geklickt werden soll, und dessen Koordinaten herausfinden lassen.

Nehmen wir an, Sie haben einen Screenshot einer *Submit*-Schaltfläche aufgenommen und in *submit.png* gespeichert. Die Funktion `locateOnScreen()` kann dann die Koordinaten der Stelle ermitteln, an der sich dieses Bild zeigt. Um das auszuprobieren, nehmen Sie einen Screenshot von einem kleinen Bereich Ihres Bildschirms auf. Speichern Sie das Bild und führen Sie folgenden Code in der interaktiven Shell aus, wobei Sie '*submit.png*' natürlich durch den Dateinamen Ihres Screenshots ersetzen müssen:

```
>>> import pyautogui  
>>> pyautogui.locateOnScreen('submit.png')  
(643, 745, 70, 29)
```

Die Funktion `locateOnScreen()` gibt ein Tupel aus vier Integerwerten zurück, nämlich der x-Koordinate des linken Rands, der y-Koordinate des oberen Rands sowie der Breite und der Höhe der ersten Stelle, an der das Motiv des Screenshots auf dem Bildschirm gefunden wurde. Auf Ihrem Computer werden sich natürlich andere Zahlenwerte ergeben als in diesem Beispiel.

Ist das, was in dem Screenshot dargestellt wird, auf dem Bildschirm nicht zu finden, gibt `locateOnScreen()` den Wert `None` zurück. Damit ein Bild erkannt werden kann, muss es mit der Darstellung auf dem Bildschirm vollständig übereinstimmen. Selbst bei einer Abweichung von nur einem Pixel gibt die Funktion `None` zurück.

Ist der Inhalt des Screenshots an mehreren Stellen auf dem Bildschirm zu finden, verwenden Sie die Funktion `locateAllOnScreen()`. Sie gibt ein Generator-Objekt zurück, das Sie an `list()` übergeben können, um eine Liste von Vier-Integer-Tupeln für jede Fundstelle zu erhalten. Setzen Sie das Beispiel in der interaktiven Shell wie folgt fort (wobei Sie '*submit.png*' wiederum durch den Dateinamen Ihres Screenshots ersetzen):

```
>>> list(pyautogui.locateAllOnScreen('submit.png'))  
[(643, 745, 70, 29), (1007, 801, 70, 29)]
```

Jedes dieser Vier-Integer-Tupel steht für einen Bereich auf dem Bildschirm. Wird das Bild nur an einer Stelle gefunden, so ergibt sich bei der Verwendung von `locateAllOnScreen()` mit `list()` eine Liste, die nur ein Tupel enthält.

Um in die Mitte des gefundenen Bereichs zu klicken, übergeben Sie das Vier-Integer-Tupel an die Funktion `center()`. Sie gibt die x- und y-Koordinaten für den Mittelpunkt des betreffenden Bereichs zurück. Probieren Sie das wie folgt in der interaktiven Shell aus, wobei Sie natürlich den Namen Ihrer Datei und die auf Ihrem Computer ermittelten Werte verwenden müssen:

```
>>> pyautogui.locateOnScreen('submit.png')
(643, 745, 70, 29)
>>> pyautogui.center((643, 745, 70, 29))
(678, 759)
>>> pyautogui.click((678, 759))
```

Wenn Sie nun die von `center()` zurückgegebenen Mittelpunktkoordinaten an `click()` übergeben, wird ein Mausklick auf der Mitte des Bildschirmbereichs ausgeführt, der mit dem an `locateOnScreen()` übergebenen Ausschnitt übereinstimmt.

## Die Tastatur steuern

PyAutoGUI verfügt auch über Funktionen, die virtuelle Tastenbetätigungen an den Computer senden. Damit können Sie Formulare ausfüllen und Text in Anwendungen eingeben.

### Strings von der Tastatur senden

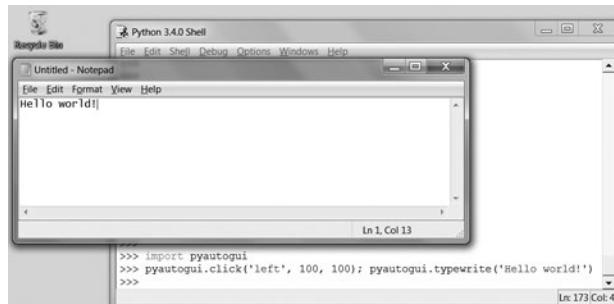
Die Funktion `pyautogui.typewrite()` sendet virtuelle Tastendrücke an den Computer. Was sie bewirken, hängt davon ab, welches Fenster und welches Textfeld den Fokus hat. Am besten sendet man zuerst einen Mausklick an das Textfeld, um sicherzugehen, dass es den Fokus hat.

Als ein einfaches Beispiel wollen wir Python den Gruß *Hello world!* in das Dateieditorfenster eingeben lassen. Platzieren Sie dazu ein neues Dateieditorfenster in der oberen linken Ecke Ihres Bildschirms, damit PyAutoGUI an die richtige Stelle klickt, um den Fokus auf dieses Fenster zu verlegen. Geben Sie dann Folgendes in die interaktive Shell ein:

```
>>> pyautogui.click(100, 100); pyautogui.typewrite('Hello world!')
```

Dadurch, dass wir die beiden Befehle durch ein Semikolon getrennt in dieselbe Zeile schreiben, halten wir die interaktive Shell davon ab, Sie nach der Ausführung der ersten Anweisung zu einer Eingabe aufzufordern. Das verhindert, dass zwischen den Aufrufen von `click()` und `typewrite()` versehentlich ein neues Fenster geöffnet wird und den Fokus erhält, was einen korrekten Ablauf des Beispiels unmöglich machen würde.

Als Erstes sendet Python einen virtuellen Mausklick an die Koordinaten (100, 100). Das entspricht einem Klick auf das Dateieditorfenster, wodurch es den Fokus erhält. Danach sendet `typewrite()` den Text *Hello world!* an das Fenster (siehe Abb. 18–3). Der Code kann also Eingaben für Sie vornehmen!



**Abb. 18–3** PyAutoGUI klickt auf das Dateieditorfenster und gibt den Text Hello world! ein.

In der Standardeinstellung gibt `typewrite()` sofort den kompletten String aus. Mit einem optionalen zweiten Argument können Sie jedoch dafür sorgen, dass hinter jedem Zeichen eine kurze Pause eingelegt wird. Dieses Argument ist ein Integer- oder Fließkommawert für die Dauer dieser Pause in Sekunden. Beispielsweise wird bei der Anweisung `pyautogui.typewrite('Hello world!', 0.25)` nach dem *H* eine Viertelsekunde gewartet, nach dem *e* eine weitere Viertelsekunde usw. Dieser »Schreibmaschinen-Effekt« kann sehr hilfreich für langsamere Anwendungen sein, die Tastatureingaben nicht schnell genug verarbeiten können, um mit PyAutoGUI Schritt zu halten.

Für Großbuchstaben und Zeichen wie ! simuliert PyAutoGUI auch die gleichzeitige Betätigung der Umschalttaste.

### Tastennamen

Nicht alle Tasten lassen sich durch einfache Textzeichen darstellen. Wie greifen Sie beispielsweise auf die Umschalttaste oder die Taste mit dem Pfeil nach links zu? In PyAutoGUI verwenden Sie für solche Tasten kurze Stringwerte, z. B. 'esc' für die **Esc**-Taste oder 'enter' für die Eingabetaste.

Statt eines einzelnen Stringarguments können Sie an `typewrite()` auch eine Liste von Tastenstrings übergeben. Im folgenden Beispiel wird erst **A** gedrückt, dann **B**, dann zweimal der Linkspfeil und schließlich **X** und **Y**:

```
>>> pyautogui.typewrite(['a', 'b', 'left', 'left', 'X', 'Y'])
```

Da der Cursor durch die Betätigung der Pfeiltaste eine Stelle nach links rückt, lautet die Ausgabe XYab. In Tabelle 18–1 finden Sie die PyAutoGUI-Tastenstrings, die Sie an `typewrite()` übergeben können, um die Betätigung von Sondertasten zu simulieren.

Um sich sämtliche Tastenstrings anzusehen, die PyAutoGUI akzeptiert, können Sie auch die Liste `pyautogui.KEYBOARD_KEYS` untersuchen. Der String 'shift' steht für die linke Umschalttaste und ist gleichbedeutend mit 'shiftleft'. Das Gleiche gilt auch für 'ctrl', 'alt' und 'win': Sie alle beziehen sich auf die jeweils linke Taste dieser Art.

String	Bedeutung
'a', 'b', 'c', 'A', 'B', 'C', '1', '2', '3', '! ', '@ ', '#' usw.	Die Tasten für die entsprechenden Buchstaben und Zeichen
'enter' (oder 'return' oder '\n')	Die Eingabetaste
'esc'	[Esc]
'shiftleft', 'shiftright'	Die linke und rechte Umschalttaste
'altright', 'altright'	Die linke und rechte [Alt]-Taste
'ctrlleft', 'ctrlright'	Die linke und rechte [Ctrl]-Taste
'tab' (oder '\t')	[Tab]
'backspace', 'delete'	[Rückschritt] und [Entf]
'pageup', 'pagedown'	[Bild aufwärts] und [Bild abwärts]
'home', 'end'	[Pos1] und [Ende]
'up', 'down', 'left', 'right'	Die Pfeiltasten nach oben, unten, links und rechts
'f1', 'f2', 'f3' usw.	Die Funktionstasten [F1] bis [F12]
'volumemute', 'volumedown', 'volumeup'	Die Tasten zum Stummschalten und zum Verringern und Erhöhen der Lautstärke. (Nicht alle Tastaturen verfügen über diese Tasten, doch das Betriebssystem versteht die simulierten Tastendrücke trotzdem.)
'pause'	[Pause]
'capslock', 'numlock', 'scrolllock'	Feststelltaste, [NumLock] und [Rollen]
'insert'	[Einfg]
'printscreen'	[Druck]
'winleft', 'winright'	Die linke und rechte Windows-Taste
'command'	Die [cmd]-Taste auf OS X (die frühere Apfel- oder Blumenkohl-Taste)
'option'	Die [alt]-Taste auf OS X (die frühere Options- oder Weiche-Taste)

**Tab. 18-1** Strings zur Darstellung von Tasten in PyAutoGUI

## Tasten drücken und loslassen

Vergleichbar mit `mouseDown()` und `mouseUp()` simulieren die Funktionen `pyautogui.keyDown()` und `pyautogui.keyUp()` das Niederdrücken bzw. Loslassen einer Taste. Als Argument wird Ihnen der String für die gewünschte Taste übergeben (siehe Tabelle 18–1). Als komfortablere Lösung bietet PyAutoGUI auch die Funktion `pyautogui.press()`, die beide Funktion kombiniert, um eine komplette Betätigung einer Taste zu simulieren.

Der folgende Code gibt ein Dollarzeichen aus, was durch Niederhalten der Umschalttaste und drücken von `[4]` erreicht wird:

```
>>> pyautogui.keyDown('shift'); pyautogui.press('4'); pyautogui.keyUp('shift')
```

Hier wird die Umschalttaste gedrückt, anschließend `[4]` gedrückt und losgelassen und schließlich die Umschalttaste losgelassen. Um einen String in ein Textfeld einzugeben, ist die Funktion `typewrite()` besser geeignet, für einzelne Tastaturbefehle an Anwendungen dagegen `press()`.

## Tastenkombinationen

Bei einer *Tastenkombination* oder einem *Tastenkürzel* drücken Sie mehrere Tasten, um eine bestimmte Funktion einer Anwendung aufzurufen. Die übliche Tastenkombination zum Kopieren der Auswahl ist `[Strg] + [C]` auf Windows und Linux bzw. `[cmd] + [C]` auf OS X. Dazu halten Sie die Taste `[Strg]` gedrückt, während Sie `[C]` drücken, und lassen dann beide Tasten los. Das können Sie mit den PyAutoGUI-Funktionen `keyDown()` und `keyUp()` wie folgt tun:

```
pyautogui.keyDown('ctrl')
pyautogui.keyDown('c')
pyautogui.keyUp('c')
pyautogui.keyUp('ctrl')
```

Das ist jedoch ziemlich umständlich. Stattdessen können Sie die Funktion `pyautogui.hotkey()` verwenden, die mehrere Tastenstrings als Argumente annimmt und die Tasten dann in der angegebenen Reihenfolge niederdrückt und anschließend in umgekehrter Reihenfolge loslässt. Damit vereinfacht sich der Code für `[Strg] + [C]` wie folgt:

```
pyautogui.hotkey('ctrl', 'c')
```

Diese Funktion ist insbesondere bei umfangreicherer Tastenkombinationen sehr praktisch. Um etwa in Word die Palette mit den Formatvorlagen anzuzeigen,

müssen Sie `[Strg] + [Alt] + [Umschalt] + [S]` drücken. Anstelle von acht Funktionsaufrufen (viermal für `keyDown()` und viermal für `keyUp()`) können Sie einfach `hotkey('ctrl', 'alt', 'shift', 's')` schreiben.

Öffnen Sie ein neues Dateieditorfenster in der linken oberen Ecke des Bildschirms und geben Sie Folgendes in die interaktive Shell ein (auf OS X müssen Sie dabei 'alt' durch 'ctrl' ersetzen):

```
>>> import pyautogui, time
>>> def commentAfterDelay():
    pyautogui.click(100, 100)      ❶
    pyautogui.typewrite('In IDLE, Alt-3 comments out a line.') ❷
    time.sleep(2)
    pyautogui.hotkey('alt', '3')   ❸

>>> commentAfterDelay()
```

Wenn Sie die hier definierte Funktion `commentAfterDelay()` aufrufen, klickt sie auf das Dateieditorfenster, um ihm den Fokus zu geben (❶), gibt den Text *In IDLE, Alt-3 comments out a line* ein (❷), wartet zwei Sekunden lang und simuliert dann die Tastenkombination `[Alt] + [3]` (bzw. `[ctrl] + [3]` auf OS X) (❸). Diese Tastenkombination fügt zwei #-Zeichen zur aktuellen Zeile hinzu und kommentiert sie damit aus. (Das ist ein praktischer Trick zum Schreiben von Code in IDLE.)

## Übersicht über die Funktionen von PyAutoGUI

Da wir in diesem Kapitel viele verschiedene Funktionen behandelt haben, gebe ich hier eine Übersicht:

- `moveTo(x, y)` Verschiebt den Mauszeiger an die angegebenen x- und y-Koordinaten.
- `moveRel(xOffset, yOffset)` Verschiebt den Mauszeiger relativ zu seiner aktuellen Position.
- `dragTo(x, y)` Verschiebt den Mauszeiger bei gedrückter linker Maustaste.
- `dragRel(xOffset, yOffset)` Verschiebt den Mauszeiger bei gedrückter linker Maustaste relativ zu seiner aktuellen Position.
- `click(x, y, button)` Simuliert einen Klick (standardmäßig mit der linken Maustaste).
- `rightClick()` Simuliert einen Rechtsklick.
- `middleClick()` Simuliert einen Klick mit der mittleren Maustaste.
- `doubleClick()` Simuliert einen Doppelklick.
- `mouseDown(x, y, button)` Simuliert das Drücken der angegebenen Maustaste an den angegebenen Koordinaten.

- `mouseUp(x, y, button)` Simuliert das Loslassen der angegebenen Maustaste an den angegebenen Koordinaten.
- `scroll(units)` Simuliert eine Betätigung des Scrollrads. Bei positiven Argumenten wird nach oben gescrollt, bei negativen nach unten.
- `typeWrite(message)` Gibt die Zeichen in dem angegebenen Nachrichtenstring aus.
- `typeWrite([key1, key2, key3])` Simuliert die Betätigung der als Tastenstrings angegebenen Tasten.
- `press(key)` Simuliert das Drücken und Loslassen der angegebenen Taste.
- `keyDown(key)` Simuliert das Niederdrücken der angegebenen Taste.
- `keyUp(key)` Simuliert das Loslassen der angegebenen Taste.
- `hotkey([key1, key2, key3])` Simuliert das Niederdrücken der angegebenen Tasten in der angegebenen Reihenfolge und das anschließende Loslassen in umgekehrter Reihenfolge.
- `screenshot()` Gibt ein Image-Objekt mit einem Screenshot zurück. (Weitere Informationen über Image-Objekte erhalten Sie in Kapitel 17.)

## Projekt: Formulare automatisch ausfüllen

Von allen langweiligen Aufgaben ist das Ausfüllen von Formularen die gefürchtetste. Wie gut, dass Sie hier im letzten Kapitel endlich ein Projekt erarbeiten werden, mit dem Sie sich dieser Pflicht entledigen können! Nehmen wir an, Sie haben eine große Menge an Daten in einem Arbeitsblatt und müssen sie mühselig in die Formularoberfläche einer anderen Anwendung eingeben – ohne dass Sie diese Arbeit einem Praktikanten aufs Auge drücken können. Einige Anwendungen verfügen zwar über eine Importfunktion, mit der Sie das Arbeitsblatt mit den Informationen laden können, aber manchmal scheint es keine andere Möglichkeit zu geben, als stundenlang stumpf vor sich hin zu klicken und zu tippen. Da Sie in diesem Buch schon so weit gekommen sind, wissen Sie bereits, dass es *natürlich* auch eine andere Möglichkeit gibt.

Für dieses Projekt verwenden wir das Google-Docs-Formular aus Abb. 18–4, das Sie von [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) herunterladen können.

The screenshot shows a web browser window displaying a Google Form titled "Generic Form". The form contains the following fields:

- A text input field labeled "Name \*".
- A text input field labeled "Greatest Fear(s)".
- A dropdown menu labeled "What is the source of your wizard powers?" with "Wand" selected.
- A question "Robocop was the greatest action movie of the 1980s." followed by a rating scale from 1 to 5, with "Strongly Agree" selected.
- A text area labeled "Additional Comments".
- A "Submit" button at the bottom.

At the bottom of the browser window, there is a note: "Never submit passwords through Google Forms."

Abb. 18–4 Das Formular für dieses Projekt

Das Programm muss folgende Dinge erledigen:

- In das erste Textfeld des Formulars klicken
- Das Formular durchgehen und dabei Informationen in jedes Feld eingeben
- Auf die *Submit*-Schaltfläche klicken
- Den Vorgang mit den nächsten Daten wiederholen

Dazu muss der Code folgende Aufgaben ausführen:

- Mit `pyautogui.click()` auf das Formular und die *Submit*-Schaltfläche klicken
- Mit `pyautogui.typewrite()` Text in die Felder eingeben
- Die Ausnahme `KeyboardInterrupt` verarbeiten, damit der Benutzer das Programm mit `Strg + C` beenden kann

Öffnen Sie ein neues Dateieditorfenster und speichern Sie das noch leere Programm als `formFiller.py`.

### Schritt 1: Den Ablauf herausfinden

Bevor Sie den Code schreiben, müssen Sie erst genau herausfinden, welche Tastendrücke und Mausklicks Sie brauchen, um das Formular einmal auszufüllen. Mit dem Skript `mouseNow.py` aus diesem Kapitel können Sie die Koordinaten für Mausklicks ermitteln. Dabei müssen Sie nur die Koordinaten des ersten Textfelds kennen, denn nachdem Sie darauf geklickt haben, können Sie anschließend `Tab` drücken,

um sich zum nächsten Feld zu bewegen. Das erspart es Ihnen, die Koordinaten sämtlicher Felder ausfindig zu machen.

Um das Formular auszufüllen, ist folgender Ablauf erforderlich:

1. Klicken Sie in das Feld *Name*. (Maximieren Sie das Browserfenster und bestimmen Sie die Koordinaten mit *mouseNow.py*. Auf OS X kann es erforderlich sein, zweimal zu klicken, nämlich einmal, um den Fokus wieder auf den Browser zu legen, und dann, um das Namensfeld anzuklicken.)
2. Geben Sie einen Namen ein und drücken Sie `Tab`.
3. Geben Sie ein, wovor Sie sich am meisten fürchten, und drücken Sie `Tab`.
4. Drücken Sie so oft die Taste mit dem abwärts weisenden Pfeil, bis Sie die richtige Quelle Ihrer magischen Kräfte ausgewählt haben: einmal für *wand*, zweimal für *amulet*, dreimal für *crystal ball* und viermal für *money*. Drücken Sie anschließend `Tab`. (Auf OS X müssen Sie die Pfeiltaste für jede Option einmal mehr drücken. Bei manchen Browsern ist es außerdem erforderlich, die Eingabetaste zu drücken.)
5. Drücken Sie die Taste mit dem nach rechts weisenden Pfeil, um die Antwort für die RoboCop-Frage auszuwählen: einmal für 2, zweimal für 3, dreimal für 4 und viermal für 5. Um die Option 1 auszuwählen, die standardmäßig markiert ist, drücken Sie einfach die Leertaste. Danach drücken Sie wieder `Tab`.
6. Geben Sie einen optionalen Kommentar ein und drücken Sie `Tab`.
7. Drücken Sie die Eingabetaste, um auf die *Submit*-Schaltfläche zu »klicken«.
8. Nach dem Absenden des Formulars wechselt der Browser zu einer Seite, auf der Sie auf einen Link klicken müssen, um wieder zu der Formularseite zurückzukehren.

Wenn Sie das Programm später erneut ausführen, kann es sein, dass Sie die Koordinaten für die Mausklicks anpassen müssen, weil das Browserfenster seine Position geändert hat. Um das zu vermeiden, sollten Sie das Browserfenster maximieren, bevor Sie die Koordinaten des ersten Formularfelds ermitteln. Es kann auch sein, dass es je nach Browser und Betriebssystem Abweichungen von dem hier beschriebenen Vorgang gibt. Bevor Sie das Programm ausführen, sollten Sie daher überprüfen, ob die Tastenkombinationen auch tatsächlich wie vorgesehen funktionieren.

## Schritt 2: Die Koordinaten ermitteln

Öffnen Sie das heruntergeladene Beispielformular (siehe Abb. 18–4) im Browser und maximieren Sie das Browserfenster. Führen Sie dann in einem neuen Terminal- oder Befehlszeilenfenster das Skript *mouseNow.py* aus und fahren Sie mit der Maus über das Feld *Name*, um dessen x- und y-Koordinaten zu ermitteln. Diese

Zahlen müssen Sie in der Variablen `nameField` des zu schreibenden Programms speichern. Bestimmen Sie auch die x- und y-Koordinaten und das RGB-Tupel der blauen *Submit*-Schaltfläche, um Sie später den Variablen `submitButton` bzw. `submitButtonColor` zuzuweisen.

Geben Sie als Nächstes einige Testdaten in das Formular ein und klicken Sie auf *Submit*. Ermitteln Sie auf der daraufhin angezeigten Seite mithilfe von `mouseNow.py` die Koordinaten des Links *Submit another response*.

Geben Sie den folgenden Quellcode für Ihr Programm ein, aber ersetzen Sie dabei alle kursiven Werte durch die Koordinaten, die Sie bei Ihren Tests ermittelt haben:

```
#! python3
# formFiller.py - Füllt das Formular automatisch aus

import pyautogui, time

# Ersetzen Sie die Koordinaten durch die auf Ihrem Computer ermittelten Werte
nameField = (648, 319)
submitButton = (651, 817)
submitButtonColor = (75, 141, 249)
submitAnotherLink = (760, 224)

# TODO: Dem Benutzer eine Möglichkeit geben, das Skript zu beenden

# TODO: Warten, bis die Formularseite geladen ist

# TODO: Das Feld Name ausfüllen

# TODO: Das Feld Greatest Fear(s) ausfüllen

# TODO: Das Feld Source of Wizard Powers ausfüllen

# TODO: Das RoboCop-Feld ausfüllen

# TODO: Das Feld Additional Comments ausfüllen

# TODO: Auf Submit klicken

# TODO: Warten, bis die Formularseite geladen ist

# TODO: Auf den Link "Submit another response" klicken
```

Jetzt brauchen Sie natürlich noch die Daten, mit denen die Formulare ausgefüllt werden sollen. In der Praxis würden solche Daten aus einem Arbeitsblatt, einer Textdatei oder einer Website kommen, weshalb Sie noch zusätzlichen Code

bräuchten, um sie in Ihr Programm zu laden. Für dieses Projekt schreiben wir die Daten einfach in eine Variable:

```
#! python3
# formFiller.py - Füllt das Formular automatisch aus

-- schnipp --

formData = [{"name": "Alice", "fear": "eavesdroppers", "source": "wand",
    "robocop": 4, "comments": "Tell Bob I said hi."},
    {"name": "Bob", "fear": "bees", "source": "amulet", "robocop": 4,
    "comments": "n/a"}, {"name": "Carol", "fear": "puppets", "source": "crystal ball",
    "robocop": 1, "comments": "Please take the puppets out of the
    break room."}, {"name": "Alex Murphy", "fear": "ED-209", "source": "money",
    "robocop": 5, "comments": "Protect the innocent. Serve the public
    trust. Uphold the law."},
]
-- schnipp --
```

Die Liste formData enthält vier Dictionarys für vier verschiedene Personen. In jedem dieser Dictionarys dienen die Namen der Textfelder als Schlüssel und die Antworten als Werte.

Die letzte vorbereitende Aufgabe besteht darin, die PyAutoGUI-Variable PAUSE so einzustellen, dass nach jedem Funktionsaufruf eine halbe Sekunde gewartet wird. Fügen Sie hinter der Zuweisung von formData noch folgende Codezeile in das Programm ein:

```
pyautogui.PAUSE = 0.5
```

### Schritt 3: Daten eingeben

Eine for-Schleife durchläuft die Dictionarys in der Liste formData und übergibt die Werte jeweils an die PyAutoGUI-Funktionen, die die Textfelder ausfüllen. Ergänzen Sie das Programm um folgenden Code:

```
#! python3
# formFiller.py - Füllt das Formular automatisch aus

-- schnipp --

for person in formData:
    # Gibt dem Benutzer eine Möglichkeit, das Skript zu beenden
    print('>>> 5 SECOND PAUSE TO LET USER PRESS CTRL-C <<<')
    time.sleep(5) ❶
```

```
# Wartet, bis die Formularseite geladen ist.  
while not pyautogui.pixelMatchesColor(submitButton[0], submitButton[1],  
submitButtonColor):    ②  
    time.sleep(0.5)  
  
-- schnipp --
```

Als kleine Sicherheitsfunktion legt das Skript Pausen von fünf Sekunden Länge ein (❶). So hat der Benutzer die Möglichkeit, **Strg** + **C** zu drücken (oder den Mauszeiger in die obere linke Ecke zu verschieben, um die Ausnahme `FailSafeException` auszulösen), damit er das Programm beenden kann, falls es irgendetwas Unerwartetes tun sollte. Das Programm wartet dann, bis die Farbe der *Submit*-Schaltfläche sichtbar wird (❷), was das Zeichen ist, dass die Formularseite vollständig geladen ist. Die Koordinaten und die Farbe dieser Schaltfläche haben Sie in Schritt 2 ermittelt und in den Variablen `submitButton` und `submitButtonColor` gespeichert. Übergeben Sie daher `submitButton[0]` und `submitButton[1]` sowie `submitButtonColor` an `pixelMatchesColor()`.

Fügen Sie hinter dem Code, der auf das Erscheinen der Farbe der *Submit*-Schaltfläche wartet, Folgendes hinzu:

```
#! python3  
# formFiller.py - Füllt das Formular automatisch aus  
  
-- schnipp --  
  
print('Entering %s info...' % (person['name'])) ❶  
pyautogui.click(nameField[0], nameField[1]) ❷  
  
# Füllt das Feld Name aus  
pyautogui.typewrite(person['name'] + '\t') ❸  
  
# Füllt das Feld Greatest Fear(s) aus  
pyautogui.typewrite(person['fear'] + '\t') ❹  
  
-- schnipp --
```

Die Aufrufe von `print()` dienen dazu, den Stand des Programms im Terminalfenster anzuzeigen, sodass der Benutzer darüber informiert ist, was gerade geschieht (❶).

Da das Programm weiß, dass das Formular geladen ist, können wir `click()` aufrufen, um in das Feld *Name* zu klicken (❷), und anschließend mit `typewrite()` den String aus `person['name']` eingeben (❸). Um das Drücken der **Tab**-Taste zu simulieren, fügen wir am Ende des an `typewrite()` übergebenen Strings das Zeichen '\t' hinzu. Dadurch wechselt der Tastaturfokus zum nächsten Feld, *Greatest Fear(s)*. Mit einem weiteren Aufruf von `typewrite()` geben wir den In-

halt von `person['fear']` in dieses Feld ein und wechseln mit dem Tabulator in das nächste Formularfeld (❸).

#### Schritt 4: Auswahllisten und Optionsschalter

Das Dropdownmenü für die Quelle der magischen Kräfte und der Optionsschalter für die RoboCop-Frage sind etwas kniffliger als die Testfelder. Wenn Sie hier mit der Maus arbeiten wollten, müssten Sie die x- und y-Koordinaten aller möglichen Antworten herausfinden. Es ist einfacher, die Pfeiltasten zu verwenden, um eine Auswahl zu treffen.

Ergänzen Sie das Programm wie folgt:

```
#! python3
# formFiller.py - Füllt das Formular automatisch aus

-- schnipp --

# Füllt das Feld Source of Wizard Powers aus
if person['source'] == 'wand':    ❶
    pyautogui.typewrite(['down', '\t'])  ❷
elif person['source'] == 'amulet':
    pyautogui.typewrite(['down', 'down', '\t'])
elif person['source'] == 'crystal ball':
    pyautogui.typewrite(['down', 'down', 'down', '\t'])
elif person['source'] == 'money':
    pyautogui.typewrite(['down', 'down', 'down', 'down', '\t'])

# Füllt das RoboCop-Feld aus
if person['robocop'] == 1:    ❸
    pyautogui.typewrite([' ', '\t'])   ❹
elif person['robocop'] == 2:
    pyautogui.typewrite(['right', '\t'])
elif person['robocop'] == 3:
    pyautogui.typewrite(['right', 'right', '\t'])
elif person['robocop'] == 4:
    pyautogui.typewrite(['right', 'right', 'right', '\t'])
elif person['robocop'] == 5:
    pyautogui.typewrite(['right', 'right', 'right', 'right', '\t'])

-- schnipp --
```

Wenn das Dropdownmenü den Fokus hat (was dadurch geschieht, dass der Code nach dem Ausfüllen des Felds *Greatest Fear(s)* die Taste `Tab` simuliert), führt eine Betätigung der Taste mit dem nach unten weisenden Pfeil dazu, dass der nächste Eintrag in der Liste ausgewählt wird. Das Programm simuliert nun so oft die Betätigung der Pfeiltaste, wie es der Wert von `person['source']` verlangt, und

wechselt danach mit `Tab` zum nächsten Feld. Wenn der Wert unter dem Schlüssel 'source' des aktuellen Dictionarys 'wand' lautet (1), simulieren wir nur eine Betätigung der Taste (um *Wand* auszuwählen) und drücken danach `Tab` (2). Für den Wert 'amulet' müssen wir ein zweifaches Drücken der Taste simulieren usw.

Die Optionsschalter für die RoboCop-Frage wählen wir mit der Taste mit dem Rechtspfeil aus. Falls die erste Option ausgewählt werden muss (3), betätigen wir einfach die Leertaste (4).

### Schritt 5: Das Formular absenden und warten

Um das Feld *Additional Comments* auszufüllen, verwenden Sie die Funktion `typewrite()`, der Sie `person['comments']` als Argument übergeben. Mit einem zusätzlichen `\t` verschieben Sie den Fokus ins nächste Feld bzw. in diesem Fall zur *Submit*-Schaltfläche. Wenn Sie nun `pyautogui.press('enter')` aufrufen, wird die Betätigung der Eingabetaste simuliert und damit das Formular abgesendet. Danach wartet das Programm fünf Sekunden lang darauf, dass die nächste Seite geladen wird.

Auf der neuen Seite gibt es den Link *Submit another response*, der im Browser eine neue, leere Formularseite anzeigt. In Schritt 2 haben Sie die Koordinaten dieses Links als Tupel in `submitAnotherLink` gespeichert, sodass Sie sie nun an `pyautogui.click()` übergeben können, um auf den Link zu klicken.

Sobald das neue Formular bereitsteht, kann die äußere for-Schleife des Skripts mit der nächsten Iteration weitermachen und die Angaben der nächsten Person in das Formular einfügen.

Vervollständigen Sie das Programm mit dem folgenden Code:

```
#! python3
# formFiller.py - Füllt das Formular automatisch aus

-- schnipp --

# Füllt das Feld Additional Comments aus
pyautogui.typewrite(person['comments'] + '\t')

# Klickt auf Submit
pyautogui.press('enter')

# Wartet, bis die Formularseite geladen ist
print('Clicked Submit.')
time.sleep(5)

# Klickt auf den Link "Submit another response"
pyautogui.click(submitAnotherLink[0], submitAnotherLink[1])
```

Wenn die Hauptschleife beendet ist, hat das Programm die Angaben aller Personen eingetragen. In diesem Beispiel waren es nur vier Personen, aber wenn es um 4000 geht, dann erspart Ihnen das Schreiben eines solchen Programms eine Menge Zeit und Tipparbeit!

## Zusammenfassung

Durch GUI-Automatisierung mithilfe des Moduls PyAutoGUI können Sie mit den Anwendungen auf Ihrem Computer arbeiten, indem Sie die Maus und die Tastatur steuern. Damit kann das Programm praktisch alles tun, was auch ein menschlicher Benutzer tun kann. Allerdings kann es dabei im Grunde genommen nicht richtig erkennen, wohin es eigentlich klickt oder wo es etwas eingibt. Daher sollten Sie dafür sorgen, dass ein GUI-Automatisierungsprogramm möglichst unverzüglich abstürzt, wenn irgendetwas schiefgeht. Ein solcher Absturz ist zwar unangenehm, aber immer noch besser, als wenn das Programm mit fehlerhaften Aktionen fortfahren würde.

Mit PyAutoGUI können Sie den Mauszeiger auf dem Bildschirm bewegen und Mausklicks, Tastendrücke und Tastenkombinationen simulieren. Das Modul kann außerdem die Farben auf dem Bildschirm untersuchen, um dem GUI-Automatisierungsprogramm eine gewisse Vorstellung von den Inhalten des Bildschirms zu geben, sodass es erkennen kann, wenn es neben der Spur liegt. Es ist sogar möglich, PyAutoGUI einen Screenshot zu übergeben und die Koordinaten des Bereichs herausfinden zu lassen, auf den geklickt werden soll.

Durch die Kombination all dieser PyAutoGUI-Funktionen können Sie stumpfsinnige, monotone Aufgaben auf Ihrem Computer automatisieren. Es hat schon fast hypnotische Wirkung, dabei zuzusehen, wie sich der Mauszeiger von selbst bewegt und Texteingaben wie von Geisterhand auf dem Bildschirm erscheinen. Es mag sich unsinnig anhören, die eingesparte Zeit damit zu verbringen, dem Programm bei der Arbeit zuzusehen, aber es ist auch durchaus befriedigend zu beobachten, wie Ihr Scharfsinn Sie davor bewahrt hat, diese langweiligen Aufgaben selbst auszuführen.

## Wiederholungsfragen

1. Wie können Sie die Sicherheitseinrichtungen von PyAutoGUI auslösen, um ein Programm anzuhalten?
2. Welche Funktion gibt die aktuelle Auflösung zurück?
3. Welche Funktion gibt die Koordinaten der aktuellen Mauszeigerposition zurück?
4. Was ist der Unterschied zwischen `pyautogui.moveTo()` und `pyautogui.moveRel()`?

5. Welche Funktionen können Sie verwenden, um mit der Maus zu ziehen?
6. Welcher Funktionsaufruf gibt die Zeichen des Strings "Hello world!" aus?
7. Wie können Sie die Betätigung von Sonderzeichen wie etwa der Taste mit dem Linkspfeil simulieren?
8. Wie können Sie den aktuellen Inhalt des Bildschirms in der Bilddatei *screen-shot.png* speichern?
9. Mit welchem Code legen Sie nach jedem Aufruf einer PyAutoGUI-Funktion eine Pause von zwei Sekunden ein?

## **Übungsprojekte**

Schreiben Sie zur Übung Programme, die folgende Aufgaben erledigen:

### **Beschäftigung vortäuschen**

Wenn sich der Mauszeiger eine bestimmte Zeit lang nicht bewegt, z. B. zehn Minuten, schließen viele Instant-Messaging-Programme daraus, dass Sie gerade unbeschäftigt sind oder nicht am Computer sitzen. Wie können Sie sich ange-sichts dessen eine Zeitlang von Ihrem Computer wegschleichen, ohne dass Ihr Instant-Messenger-Status als »unbeschäftigt« angezeigt wird? Schreiben Sie ein Skript, das den Mauszeiger alle zehn Sekunden anstößt. Der Mauszeiger soll sich dabei nur ganz geringfügig bewegen, sodass dieser Vorgang nicht stört, wenn Sie den Computer verwenden müssen, während das Skript läuft.

### **Instant-Messenger-Bot**

Google Talk, Skype, Yahoo Messenger, AIM und andere Instant-Messaging-Anwendungen verwenden oft firmeneigene Protokolle, die es schwer machen, Python-Module für den Umgang mit diesen Programmen zu schreiben. Allerdings können diese Protokolle Sie nicht davon abhalten, ein GUI-Automatisierungswerkzeug zu schreiben.

In der Suchleiste von Google Talk können Sie einen Benutzernamen von Ihrer Freundesliste eingeben. Wenn Sie dann die Eingabetaste drücken, wird ein Chatfenster geöffnet, das automatisch den Fokus erhält. Bei anderen Instant-Messaging-Anwendungen gibt es ähnliche Möglichkeiten, um ein neues Chatfenster zu öffnen. Schreiben Sie ein Programm, das automatisch eine Benachrichtigung an eine ausgewählte Gruppe von Personen auf Ihrer Freundesliste sendet. Dieses Programm muss mit verschiedenen besonderen Umständen umgehen können, z. B. damit, dass Freunde offline sind, dass das Chatfenster an unterschiedlichen Koordinaten auf dem Bildschirm erscheint oder dass Bestäti-

gungsdialogfelder den Benachrichtigungsvorgang unterbrechen. Das Programm muss die Interaktion mit der GUI anhand von Screenshots steuern und erkennen können, ob seine virtuellen Tastenbetätigungen gesendet werden oder nicht.

### Hinweis

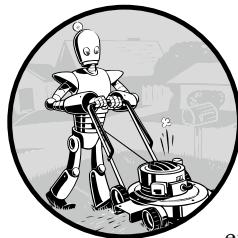
Damit Sie beim Schreiben dieses Programms nicht Ihre echten Freunde mit unsinnigen Nachrichten zumüllen, sollten Sie vorab einige Testkonten anlegen.

### Tutorial für einen Spiele-Bot

Auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren) finden Sie einen Link zu dem großartigen Tutorial »How to Build a Python Bot That Can Play Web Games«. Darin wird erklärt, wie Sie in Python ein GUI-Automatisierungsprogramm schreiben, das in der Lage ist, das Flash-Spiel Sushi Go Round zu spielen. Bei diesem Spiel müssen Sie auf die Schaltflächen für die richtigen Zutaten klicken, um Sushi-Bestellungen zusammenzustellen. Je schneller Sie das machen, ohne dass Ihnen dabei Fehler unterlaufen, umso mehr Punkte erhalten Sie. Diese Aufgabe ist ideal für ein GUI-Automatisierungsprogramm geeignet (mit dem Sie sich einen High-score erschwindeln können). In dem Tutorial werden viele der Themen aus diesem Kapitel besprochen, aber auch die grundlegenden Bilderkennungsfunktionen von PyAutoGUI.

# A

## Drittanbietermodule installieren



Neben der Standardbibliothek der Module, die im Lieferumfang von Python enthalten sind, gibt es noch Module, die andere Entwickler geschrieben haben, um die Möglichkeiten von Python zu erweitern. Das wichtigste Werkzeug zur Installation solcher Drittanbietermodule ist das Python-Tool Pip. Es lädt Module von <https://pypi.python.org/>, der Website der Python Software Foundation, sicher auf Ihren Computer herunter und installiert sie. PyPI, der Python Package Index, ist eine Art kostenloser »App-Store« für Python-Module.

### Pip

Die ausführbare Datei von Pip heißt auf Windows `pip` und auf OS X und Linux `pip3`. Zu finden ist sie in `C:\Python34\Scripts\pip.exe` (Windows), `/Library/Frameworks/Python.framework/Versions/3.4/bin/pip3` (OS X) bzw. `/usr/bin/pip3` (Linux).

Auf Windows und OS X wird Pip automatisch zusammen mit Python 3.4 installiert. Unter Linux ist eine separate Installation erforderlich. Öffnen Sie

dazu ein Terminalfenster und geben Sie auf Ubuntu und Debian `sudo apt-get install python3-pip` ein, auf Fedora `sudo yum install python3 -pip`. Um die Software installieren zu können, müssen Sie das Passwort des Computeradministrators eingeben.

## Drittanbietermodule installieren

Pip ist zur Ausführung an der Befehlszeile gedacht: Sie geben den Befehl `install` und den Namen des gewünschten Moduls an. Auf Windows schreiben Sie dazu einfach `pip install Modulname`. Auf OS X und Linux müssen Sie `pip3` mit dem Präfix `sudo` ausführen, um Administratorrechte für die Installation des Moduls bereitzustellen: `sudo pip3 install Modulname`.

Wenn ein Modul bereits installiert ist und Sie es auf die neueste Version aktualisieren möchten, die auf PyPI verfügbar ist, führen Sie `pip install -U Modulname` aus (bzw. `pip3 install -U Modulname` auf OS X und Linux).

Um zu prüfen, ob das Modul korrekt installiert wurde, führen Sie in der interaktiven Shell `import Modulname` aus. Wenn keine Fehlermeldungen angezeigt werden, können Sie davon ausgehen, dass die Installation erfolgreich verlaufen ist.

Um die in diesem Buch besprochenen Module zu installieren, führen Sie die folgenden Befehle aus (denken Sie daran, dass Sie auf OS X und Linux `pip3` statt `pip` schreiben müssen):

- `pip install send2trash`
- `pip install requests`
- `pip install beautifulsoup4`
- `pip install selenium`
- `pip install openpyxl`
- `pip install PyPDF2`
- `pip install python-docx (install python-docx, nicht docx)`
- `pip install imapclient`
- `pip install pyzmail`
- `pip install twilio`
- `pip install pillow`
- `pip install pyobjc-core (nur auf OS X)`
- `pip install pyobjc (nur auf OS X)`
- `pip install python3-xlib (nur auf Linux)`
- `pip install pyautogui`

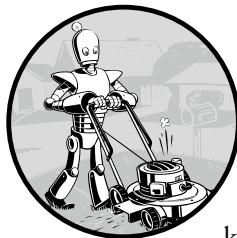
**Hinweis**

Für OS X-Benutzer: Die Installation des Moduls pyobjc kann 20 Minuten und länger dauern. Werden Sie also nicht nervös, wenn Sie etwas länger warten müssen. Um die Gesamtinstallationszeit zu verringern, sollten Sie als Erstes pyobjc-core installieren.



# B

## Programme ausführen



Wenn im Dateieditor von IDLE ein Programm geöffnet ist, können Sie es einfach dadurch ausführen, dass Sie **F5** drücken oder im Menü *Run > Run Module* wählen. Das ist eine praktische Möglichkeit, um Programme auszuführen, während Sie sie schreiben, aber es wäre ziemlich umständlich, immer erst IDLE öffnen zu müssen, um ein fertiges Programm zu starten. Dazu gibt es komfortablere Möglichkeiten.

### Die Shebang-Zeile

Die erste Zeile eines Python-Programms sollte die *Shebang-Zeile* sein, die dem Computer mitteilt, dass Python das Programm ausführen soll. Sie beginnt mit der Zeichenfolge `#!`, der Rest allerdings hängt vom Betriebssystem ab:

- Auf Windows lautet die Shebang-Zeile `#! python3`.
- Auf OS X lautet die Shebang-Zeile `#! /usr/bin/env python3`.
- Auf Linux lautet die Shebang-Zeile `#! /usr/bin/ python3`.

Von IDLE aus können Sie Python-Skripte auch ohne Shebang-Zeile ausführen. Um sie von der Befehlszeile aus zu starten, ist diese Zeile jedoch notwendig.

## Python-Programme unter Windows ausführen

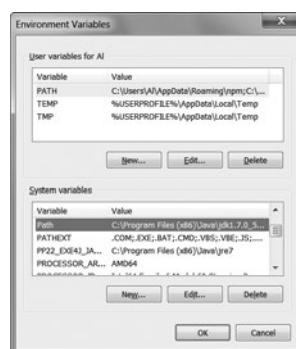
Unter Windows befindet sich der Python-3.4-Interpreter in `C:\Python34\python.exe`. Alternativ können Sie auch das komfortable Programm `py.exe` verwenden, das die Shebang-Zeile zu Beginn des Quellcodes einer `.py`-Datei liest und die passende Python-Version für das Skript ausführt, falls mehrere Versionen auf dem Computer installiert sind.

Um ein Python-Programm bequem ausführen zu können, legen Sie eine *Batchdatei* (mit der Endung `.bat`) dafür an. Dazu erstellen Sie eine neue Textdatei, die nur eine einzige Zeile enthält:

```
@py.exe C:\Pfad_zu_Ihrem\pythonSkript.py %*
```

Ersetzen Sie den Pfad durch den absoluten Pfad zu Ihrem Programm und speichern Sie die Datei mit der Endung `.bat` (z. B. als `pythonScript.bat`). Diese Batchdatei erspart es Ihnen, jedes Mal den kompletten absoluten Pfad zu dem Python-Programm einzugeben, wenn Sie es ausführen wollen. Ich rate Ihnen, alle Batch- und `.py`-Dateien in einem einzigen Ordner unterzubringen, z. B. `C:\MyPythonScripts` oder `C:\Users\IhrName\PythonScripts`.

Fügen Sie diesen Ordner zu dem Systempfad von Windows hinzu, sodass Sie die darin enthaltenen Batchdateien über das *Ausführen*-Dialogfeld starten können. Dazu müssen Sie die Umgebungsvariable `PATH` ändern. Klicken Sie auf *Start* und geben Sie *Umgebungsvariablen für dieses Konto bearbeiten* ein. Die Option sollte schon nach den ersten eingegebenen Zeichen vervollständigt werden. Wenn Sie darauf klicken, wird das Fenster *Umgebungsvariablen* aus Abb. B.1 geöffnet.



**Abb. B-1** Abbildung B.1: Das Fenster Umgebungsvariablen von Windows

Markieren Sie unter *Systemvariablen* den Eintrag *Path* und klicken Sie auf *Bearbeiten*. Im Textfeld *Feld* geben Sie C:\MyPythonScripts ein und klicken auf OK. Jetzt können Sie alle Python-Skripte in C:\MyPythonScripts ausführen, indem Sie einfach **Windows** + **R** drücken und den Namen des Skripts eingeben. Durch die Eingabe von **pythonScript** beispielsweise starten Sie *pythonScript.bat*. Das wiederum erspart es Ihnen, im Dialogfeld *Ausführen* den kompletten Befehl py.exe C:\MyPythonScripts\pythonScript.py einzugeben.

## Python-Programme unter OS X und Linux ausführen

Wenn Sie auf OS X *Programme > Dienstprogramme > Terminal* wählen, erscheint ein Terminalfenster. Darin können Sie Befehle als Text eingeben, also ohne sich durch eine grafische Benutzeroberfläche zu klicken. Um auf Ubuntu ein Terminalfenster zu öffnen, drücken Sie die Taste **Win** oder **Super**, um das Dashboard zu öffnen, und geben **Terminal** ein.

Wenn Sie das Terminalfenster aufrufen, wirken sich die eingegebenen Befehle zunächst auf den Benutzerordner Ihres Kontos aus. Mit dem Benutzernamen *asweigart* ist mein Benutzerordner auf OS X /Users/asweigart und auf Linux /home/asweigart. Als Abkürzung für den Benutzerordner können Sie auch eine Tilde (~) verwenden. Um zu Ihrem Benutzerordner zu gelangen, können Sie also ganz einfach cd ~ eingeben. Mit dem Befehl cd lässt sich jedes beliebige Verzeichnis zum aktuellen Arbeitsverzeichnis machen. Sowohl auf OS X als auch auf Linux können Sie sich mit dem Befehl pwd anzeigen lassen, was das aktuelle Arbeitsverzeichnis ist.

Um Python-Programme auszuführen, speichern Sie die *py*-Dateien in Ihrem Benutzerordner. Ändern Sie dann die Berechtigungen für diese Dateien, sodass sie ausführbar sind. Dazu führen Sie chmod +x pythonScript.py aus. Dateiberechtigungen zu erklären, würde in diesem Buch zu weit führen. Merken Sie sich nur diesen Befehl, den Sie ausführen müssen, damit Sie das betreffende Programm vom Terminalfenster aus starten können. Anschließend können Sie jederzeit in einem Terminalfenster ./pythonScript.py eingeben. Die Shebang-Zeile am Anfang des Skripts teilt dem Betriebssystem mit, wo es den Python-Interpreter findet.

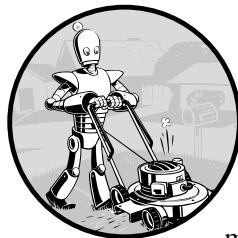
## Python-Programme mit ausgeschalteten Zusicherungen ausführen

Sie können eine leichte Leistungssteigerung bewirken, indem Sie assert-Anweisungen in Ihren Python-Programmen ausschalten. Wenn Sie Python im Terminal ausführen, geben Sie dazu den Schalter -O hinter python bzw. python3 und vor dem Namen der .py-Datei an. Dadurch wird eine optimierte Version des Programms ausgeführt, bei der keine Überprüfung der Zusicherungen stattfindet.



# C

## Antworten auf die Wiederholungsfragen



Dieser Anhang enthält die Antworten auf die Wiederholungsfragen am Ende der einzelnen Kapitel. Ich rate Ihnen sehr dazu, sich die Zeit zu nehmen und diese Fragen durchzuarbeiten. Programmieren erfordert mehr, als die Syntax und die Funktionsnamen auswendig zu lernen. Wie beim Erlernen einer Fremdsprache fahren Sie umso besser, je mehr Übung Sie haben. Es gibt auch viele Websites, auf denen Sie praktische Programmierübungen finden. Eine Liste solcher Websites erhalten Sie auf [www.dpunkt.de/automatisieren](http://www.dpunkt.de/automatisieren).

### Kapitel 1

1. Die Operatoren sind +, -, \* und /, die Werte 'hello', -88.8 und 5.
2. Der String ist 'spam', die Variable spam. Strings beginnen und enden jeweils mit Anführungszeichen.
3. Die in diesem Kapitel eingeführten Datentypen sind Integer, Fließkommazahlen und Strings.

4. Ein Ausdruck ist eine Kombination aus Werten und Operatoren. Alle Ausdrücke können zu einem einzelnen Wert ausgewertet (reduziert) werden.
5. Ein Ausdruck wird zu einem einzelnen Wert ausgewertet, eine Anweisung nicht.
6. Die Variable bacon ist auf 20 gesetzt. Der Ausdruck bacon + 1 weist bacon keinen neuen Wert zu. (Dazu bräuchten Sie die Zuweisungsanweisung bacon = bacon + 1.)
7. Beide Ausdrücke werden zu dem String 'spamspamspam' ausgewertet.
8. Variablennamen können nicht mit einer Zahl beginnen.
9. Die Funktionen int(), float() und str() werden zu der Integer-, Fließkomma- bzw. Stringversion des übergebenen Werts ausgewertet.
10. Dieser Ausdruck ruft einen Fehler hervor, da 99 ein Integer ist und Strings nur mit dem Operator + mit anderen Strings verkettet werden können. Richtig wäre 'I have eaten ' + str(99) + ' burritos.'.

## Kapitel 2

1. True und False mit großem T bzw. F und dem Rest in Kleinbuchstaben
2. and, or und not
3. True and True ist True.  
True and False ist False.  
False and True ist False.  
False and False ist False.  
True or True ist True.  
True or False ist True.  
False or True ist True.  
False or False ist False.  
not True ist False.  
not False ist True.
4. False  
False  
True  
False  
False  
True
5. ==, !=, <, >, <= und >=
6. == ist der Gleichheitsoperator, der zwei Werte auf Gleichheit prüft und einen booleschen Wert ergibt; = ist der Zuweisungsoperator, der einen Wert in einer Variablen speichert.

7. Eine Bedingung ist ein Ausdruck in einer Flusssteuerungsanweisung, der zu einem booleschen Wert ausgewertet wird.
8. Die drei Blöcke sind der gesamte Inhalt der if-Anweisung und die Zeilen `print('bacon')` und `print('ham')`.

```
print('eggs')
if spam > 5:
    print('bacon')
else:
    print('ham')
print('spam')
```

9. Der Code:

```
if spam == 1:
    print('Hello')
elif spam == 2:
    print('Howdy')
else:
    print('Greetings!')
```

10. Um ein Programm zu beenden, das in einer Endlosschleife gefangen ist, drücken Sie **[Strg] + [C]**.
11. Die Anweisung `break` setzt die Programmausführung unmittelbar hinter der Schleife fort. Bei `continue` fährt die Programmausführung mit dem Beginn der Schleife fort.
12. Alle Angaben bewirken das Gleiche. Bei `range(10)` ergibt sich ein Bereich von 0 bis (ausschließlich) 10, bei `range(0, 10)` wird der Beginn des Bereichs ausdrücklich auf 0 festgelegt und bei `range(0, 10, 1)` ausdrücklich angegeben, dass die Variable bei jedem Schleifendurchlauf um 1 erhöht werden soll.
13. Der Code:

```
for i in range(1, 11):
    print(i)
```

und:

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

14. Diese Funktion kann mit `spam.bacon()` aufgerufen werden.

## Kapitel 3

1. Durch den Einsatz von Funktionen können Sie doppelten Code vermeiden. Dadurch werden Ihre Programme kürzer und leichter lesbar und lassen sich einfacher ändern.
2. Der Code in einer Funktion wird beim Aufruf der Funktion ausgeführt, nicht bei ihrer Definition.
3. Die Anweisung `def` definiert (d. h. erstellt) eine Funktion.
4. Eine Funktion besteht aus der Anweisung `def` und dem Code in der `def`-Klausel. Ein Funktionsaufruf verschiebt die Programmausführung zu dieser Funktion und wird zum Rückgabewert der Funktion ausgewertet.
5. Es gibt einen globalen Gültigkeitsbereich. Bei jedem Funktionsaufruf wird ein lokaler Gültigkeitsbereich erstellt.
6. Wenn eine Funktion die Steuerung zurückgibt, wird ihr lokaler Gültigkeitsbereich zerstört und alle darin befindlichen Variablen werden vergessen.
7. Der Rückgabewert ist der Wert, zu dem eine Funktion ausgewertet wird. Wie jeder andere Wert kann auch ein solcher Rückgabewert als Teil eines Ausdrucks verwendet werden.
8. Wenn eine Funktion keine `return`-Anweisung hat, ist ihr Rückgabewert `None`.
9. Die Anweisung `global` sorgt dafür, dass sich die Verwendung einer Variablen in einer Funktion auf die globale Variable dieses Namens bezieht.
10. Der Datentyp von `None` ist `NoneType`.
11. Diese `import`-Anweisung importiert das Modul `areallyyourpetsnamederic` (wobei es ein solches Python-Modul übrigens nicht gibt).
12. Diese Funktion kann mit `spam.bacon()` aufgerufen werden.
13. Stellen Sie die Codezeile, die den Fehler verursachen kann, in eine `try`-Klausel.
14. Der Code, der einen Fehler hervorrufen kann, gehört in die `try`-Klausel, der Code, der bei einem Fehler ausgeführt werden soll, in die `except`-Klausel.

## Kapitel 4

1. Der leere Listenwert, also ein Listenwert, der keine Einträge enthält. Dies ähnelt dem leeren Stringwert ''.
2. `spam[2] = 'hello'` (Beachten Sie, dass sich der dritte Wert in einer Liste an der Indexposition 2 befindet, da die Zählung im Index bei 0 beginnt.)
3. 'd' (Beachten Sie, dass '3' \* 2 den String '33' ergibt, der vor der Division durch 11 erst an `int()` übergeben werden muss. Daraus ergibt sich schließlich 3. Ausdrücke können überall dort verwendet werden, wo auch Werte zulässig sind.)

4. 'd' (Negative Indizes werden vom Ende ausgehend gezählt.)
5. ['a', 'b']
6. 1
7. [3.14, 'cat', 11, 'cat', True, 99]
8. [3.14, 11, 'cat', True]
9. Der Operator für die Listenverkettung ist \*, der Operator für die Replikation ist \* (wie bei Strings).
10. `append()` hängt Werte nur am Ende einer Liste an, `insert()` dagegen kann sie an beliebiger Stelle in der Liste platzieren.
11. Die Anweisung `del` und die Listenmethode `remove()` bilden zwei Möglichkeiten, um Werte von einer Liste zu entfernen.
12. Sowohl Listen als auch Werte können an `len()` übergeben werden, haben Indizes und Slices, lassen sich in for-Schleifen verwenden, verketten und wiederholen und zusammen mit den Operatoren `in` und `not in` einsetzen.
13. Listen sind veränderbar. Sie können ihnen Werte hinzufügen, Werte entfernen und ändern. Tupel dagegen sind unverändert. Außerdem werden Tupel mit runden Klammern geschrieben, Listen dagegen mit eckigen.
14. (42,) (Das nachfolgende Komma ist unverzichtbar.)
15. Die Funktionen `tuple()` bzw. `list()`.
16. Sie enthalten Verweise auf Listenwerte.
17. Die Funktion `copy.copy()` erstellt eine flache Kopie einer Liste, `copy.deepcopy()` dagegen eine tiefe Kopie, bei der auch alle Listen innerhalb der Liste kopiert werden.

## Kapitel 5

1. Zwei geschweifte Klammern: {}
2. {'foo': 42}
3. In einem Dictionary sind die Einträge ungeordnet, in einer Liste geordnet.
4. Sie erhalten den Fehler `KeyError`.
5. Es gibt keinen Unterschied. Der Operator `in` prüft, ob der Wert als Schlüssel in dem Dictionary vorhanden ist.
6. 'cat' in `spam` prüft, ob es den Schlüssel 'cat' in dem Dictionary gibt, 'cat' in `spam.values()` dagegen, ob es unter einem der Schlüssel von `spam` den Wert 'cat' gibt.
7. `spam.setdefault('color', 'black')`
8. `pprint.pprint()#`

## Kapitel 6

1. Maskierungssequenzen stehen für Zeichen in Stringvalues, die sich auf andere Weise nur schwer oder nicht im Code wiedergeben lassen.
2. \n ist ein Zeilenumbruch, \t ein Tabulator.
3. Die Maskierungssequenz \\ steht für einen Backslash.
4. Das einfache Anführungszeichen in Howl's ist unproblematisch, da Anfang und Ende des Strings durch doppelte Anführungszeichen gekennzeichnet sind.
5. In mehrzeiligen Strings können Sie Zeilenumbrüche verwenden, ohne auf die Maskierungssequenz \n zurückgreifen zu müssen.
6. Die Ausdrücke werden wie folgt ausgewertet:

- 'e'
- 'Hello'
- 'Hello'
- 'lo world!'

7. Die Ausdrücke werden wie folgt ausgewertet:

- 'HELLO'
- True
- 'hello'

8. Die Ausdrücke werden wie folgt ausgewertet:

- ['Remember,', 'remember,', 'the', 'fifth', 'of', 'November. ']
- 'There-can-be-only-one.'

9. Die Stringmethoden rjust(), ljust() bzw. center()

10. Die Methoden lstrip() und rstrip() entfernen Weißraum vom linken bzw. rechten Ende des Strings.

## Kapitel 7

1. Die Funktion re.compile() gibt Regex-Objekte zurück.
2. Bei der Verwendung von Rohstrings müssen Backslashes nicht maskiert werden.
3. Die Methode search() gibt Match-Objekte zurück.
4. Die Methode group() gibt die Strings des übereinstimmenden Textes zurück.
5. Gruppe 0 ist die gesamte Übereinstimmung, Gruppe 1 enthält den ersten Satz von Klammern, Gruppe 2 den zweiten.

6. Punkte und Klammern können mit Backslashes maskiert werden: \., \( und \).
7. Weist der reguläre Ausdruck keine Gruppen auf, so wird eine Liste von Strings zurückgegeben, anderenfalls eine Liste von Tupeln von Strings.
8. Das Zeichen | bedeutet, dass eine Übereinstimmung entweder mit der einen oder mit der anderen Gruppe gefunden werden muss.
9. Das Zeichen ? bedeutet »null oder ein Vorkommen der vorausgehenden Gruppe« oder kennzeichnet die nicht gierige Suche nach Übereinstimmungen.
10. Das Zeichen + steht für ein oder mehr Vorkommen, \* für null oder mehr.
11. Das Zeichen {3} steht für genau drei Vorkommen der vorausgehenden Gruppe, {3, 5} für drei bis fünf Vorkommen.
12. Die Klassenkurzschreibweisen \d, \w und \s stehen für eine einzelne Ziffer, ein Wort bzw. ein Weißraumzeichen.
13. Die Klassenkurzschreibweisen \D, \W und \S stehen für ein einzelnes Zeichen, das keine Ziffer, kein Wort bzw. kein Weißraumzeichen ist.
14. Wenn Sie re.I oder re.IGNORECASE als zweites Argument an re.compile() übergeben, wird bei der Suche nicht zwischen Groß- und Kleinschreibung unterschieden.
15. Das Zeichen . steht normalerweise für jedes Zeichen außer dem Zeilenumbruch. Wenn Sie re.DOTALL als zweites Argument an re.compile() übergeben, werden auch Zeilenumbrüche als Übereinstimmungen mit dem Punkt gewertet.
16. Mit .\* wird eine gierige Suche durchgeführt, mit .\*? eine nicht gierige.
17. Entweder [0-9a-z] oder [a-z0-9]
18. 'X drummers, X pipers, five rings, X hens'
19. Mit dem Argument re.VERBOSE können Sie dem an re.compile() übergebenen String auch Weißraumzeichen und Kommentare hinzufügen.
20. re.compile(r'^\d{1,3}(,{3})\*\$') erstellt diesen regulären Ausdruck, aber andere Strings können ähnliche reguläre Ausdrücke hervorrufen.
21. re.compile(r'[A-Z][a-z]\*\sNakamoto')
22. re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re.IGNORECASE)

## Kapitel 8

1. Relative Pfade sind relativ zum aktuellen Arbeitsverzeichnis.
2. Absolute Pfade beginnen beim Wurzel- oder Stammordner, z. B. / oder C:\.
3. Die Funktion os.getcwd() gibt das aktuelle Arbeitsverzeichnis zurück, die Funktion os.chdir() ändert es.
4. Der Ordner . ist der aktuelle Ordner und .. der übergeordnete Ordner.

5. C:\bacon\eggs ist der Verzeichnisname, *spam.txt* der Grundname.
6. Der String 'r' steht für den Lesemodus, 'w' für den Schreibmodus und 'a' für den Anhangemodus.
7. Eine im Schreibmodus geöffnete, bereits vorhandene Datei wird komplett überschrieben.
8. Die Methode `read()` gibt den gesamten Inhalt der Datei als einzelnen Stringwert zurück. `readlines()` dagegen gibt eine Liste von Strings für die einzelnen Zeilen in der Datei zurück.
9. Ein Shelf-Wert ähnelt einem Dictionary-Wert: Er verfügt über Schlüssel und Werte und die Methoden `keys()` und `values()`, die ähnlich wie die gleichnamigen Dictionary-Methoden funktionieren.

## Kapitel 9

1. Die Funktion `shutil.copy()` kopiert eine einzelne Datei, `shutil.copytree()` dagegen einen gesamten Ordner mit allen seinen Inhalten.
2. Die Funktion `shutil.move()` wird zum Umbenennen und zum Verschieben von Dateien verwendet.
3. Die `send2trash`-Funktionen verschieben Dateien oder Ordner in den Papierkorb, die `shutil`-Funktionen löschen sie endgültig.
4. Die Funktion `zipfile.Zipfile()` entspricht der Funktion `open()`. Das erste Argument ist der Dateiname, das zweite der Modus, in dem die Zip-Datei geöffnet werden soll (Lesen, Schreiben oder Anhängen).

## Kapitel 10

1. `assert(spam >= 10, 'The spam variable is less than 10.')`
2. `assert(eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!')` oder `assert(eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!')`
3. `assert(False, 'This assertion always triggers.')`
4. Um `logging.debug()` aufrufen zu können, müssen am Anfang Ihres Programms die folgenden beiden Zeilen stehen:

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s
- %(message)s')
```

5. Um mit `logging.debug()` Protokollnachrichten an die Datei `programLog.txt` senden zu können, müssen am Anfang Ihres Programms die folgenden beiden Zeilen stehen:

```
import logging
logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')
```

6. DEBUG, INFO, WARNING, ERROR und CRITICAL
7. `logging.disable(logging.CRITICAL)`
8. Sie können Protokolliermeldungen abschalten, ohne die Aufrufe der Protokollfunktionen entfernen zu müssen. Es ist auch möglich, gezielt die Protokolliermeldungen niedrigerer Schweregrade abzuschalten. Protokolliermeldungen tragen einen Zeitstempel.
9. Mit der Schaltfläche *Step* steigt der Debugger in einen Funktionsaufruf ein. Mit *Over* wird die Funktion schnell ausgeführt, ohne in sie einzusteigen. *Out* führt den Rest des Codes in der Funktion aus, in der sich der Debugger gerade befindet, und verlässt sie dann.
10. Nach dem Klick auf *Go* hält der Debugger an, wenn er das Ende des Programms oder einen Haltepunkt erreicht hat.
11. Mit einem Haltepunkt legen Sie fest, bei welcher Codezeile der Debugger die Programmausführung anhalten soll.
12. Um in IDLE einen Haltepunkt zu setzen, rechtsklicken Sie auf die betreffende Zeile und wählen *Set Breakpoint* im Kontextmenü.

## Kapitel 11

1. Die Methode `open()` des Moduls `webbrowser()` startet lediglich einen Webbrowser und öffnet darin den angegebenen URL. Das Modul `requests` kann Dateien und Seiten aus dem Web herunterladen, das Modul `Beautiful Soup` analysiert den HTML-Text und das Modul `Selenium` kann einen Browser starten und steuern.
2. Die Funktion `requests.get()` gibt ein `Response`-Objekt zurück, dessen `text`-Attribut den heruntergeladenen Inhalt als String enthält.
3. Die Methode `raise_for_status()` löst eine Ausnahme aus, wenn es bei dem Download zu einem Problem gekommen ist. Bei einem erfolgreichen Download tut es nichts.
4. Das Attribut `status_code` des `Response`-Objekts enthält den HTTP-Statuscode.

5. Nachdem Sie auf Ihrem Computer eine neue Datei im binären Schreibmodus ('wb') geöffnet haben, durchlaufen Sie in einer for-Schlaufe die Methode `iter_content()` des Response-Objekts, um dessen Inhalt abschnittsweise in die Datei zu schreiben, wie das folgende Beispiel zeigt:

```
saveFile = open('filename.html', 'wb')
for chunk in res.iter_content(100000):
    saveFile.write(chunk)
```

6. Um die Entwickertools zu öffnen, drücken Sie in Chrome **F12** und in Firefox **Strg** + **Umschalt** + **C** (Windows und Linux) bzw. **cmd** + **alt** + **C** (OS X).
7. Rechtsklicken Sie auf das Element auf der Seite und wählen Sie im Kontextmenü *Element untersuchen*.
8. '#main'
9. '.highlight'
10. 'div div'
11. 'button[value="favorite"]'
12. spam.getText()
13. linkElem.attrs
14. Das Modul Selenium importieren Sie mit `from selenium import webdriver`.
15. Die `find_element_*`-Methoden geben das erste übereinstimmende Element als WebElement-Objekt zurück, die `find_elements_*`-Methoden eine Liste aller übereinstimmenden Objekte, ebenfalls als WebElement-Objekte.
16. Die Methoden `click()` und `send_keys()` simulieren Mausklicks bzw. Tastenbe-tätigungen.
17. Wenn Sie in einem Formular die Methode `submit()` aufrufen, senden Sie das Formular damit ab.
18. Die WebDriver-Methoden `forward()`, `back()` und `refresh()` simulieren die Browserschaltflächen *Vorwärts*, *Zurück* und *Aktualisieren*.

## Kapitel 12

1. Die Funktion `openpyxl.load_workbook()` gibt ein Workbook-Objekt zurück.
2. Die Funktion `get_sheet_names()` gibt ein Worksheet-Objekt zurück.
3. Rufen Sie `wb.get_sheet_by_name('Sheet1')` auf.
4. Rufen Sie `wb.get_active_sheet ()` auf.
5. `sheet['C5'].value` oder `sheet.cell(row=5, column=3).value`
6. `sheet['C5'] = 'Hello'` oder `sheet.cell(row=5, column=3).value = 'Hello'`

7. `cell.row` und `cell.column`
8. Sie geben den höchsten Spalten- bzw. Zeilenwert in dem Arbeitsblatt als Integerwert zurück.
9. `openpyxl.cell.column_index_from_string('M')`
10. `openpyxl.cell.get_column_letter(14)`
11. `sheet['A1':'F1']`
12. `wb.save('example.xlsx')`
13. Eine Formel wird wie jeder andere Wert angegeben. Setzen Sie das Attribut `value` der Zelle auf den String mit dem Formeltext. Denken Sie daran, dass Formeln mit dem Zeichen = beginnen.
14. Übergeben Sie beim Aufruf von `load_workbook()` den Wert `True` für das Schlüsselwortargument `data_only`.
15. `sheet.row_dimensions[5].height = 100`
16. `sheet.column_dimensions['C'].hidden = True`
17. OpenPyXL 2.0.5 kann keine Bereiche fixieren und keine Titel, Bilder und Diagramme ausgeben.
18. Fixierte Zeilen und Spalten bleiben immer auf dem Bildschirm sichtbar. Sie sind gut als Zeilen- bzw. Spaltenköpfe geeignet.
19. `openpyxl.charts.Reference()`, `openpyxl.charts.Series()`, `openpyxl.charts.BarChart()`, `chartObj.append(seriesObj)`, und `add_chart()`

## Kapitel 13

1. Ein von `open()` zurückgegebenes File-Objekt
2. Binärer Lesemodus ('rb') für `PdfFileReader()` und binärer Schreibmodus ('wb') für `PdfFileWriter()`
3. Der Aufruf von `getPage(4)` gibt ein Page-Objekt für Seite 5 zurück, da die erste Seite als Seite 0 gezählt wird.
4. Die Variable `numPages` enthält einen Integerwert mit der Anzahl der Seiten im `PdfFileReader`-Objekt.
5. Rufen Sie `decrypt('swordfish')` auf.
6. Die Methoden `rotateClockwise()` und `rotateCounterClockwise()`. Der Drehwinkel wird als Integerargument übergeben.
7. `docx.Document('demo.docx')`
8. Ein Dokument enthält mehrere Absätze. Ein Absatz beginnt auf einer neuen Zeile und umfasst mehrere »Runs«. Runs sind durchgängige Zeichengruppen in einem Absatz.
9. Verwenden Sie `doc.paragraphs`.

10. Ein Run-Objekt hat diese Variablen (nicht ein Paragraph-Objekt).
11. Mit `True` wird das Run-Objekt immer fett formatiert und mit `False` immer nicht fett, und zwar unabhängig von der Einstellung für Fettschrift des Formats. Bei `None` erhält das Run-Objekt die Einstellung für Fettschrift aus dem Format.
12. Rücken Sie die Funktion `docx.Document()` auf.
13. `doc.add_paragraph('Hello there!')`
14. Die Integer 0, 1, 2, 3 und 4

## Kapitel 14

1. Excel-Arbeitsblätter können Werte mit anderen Datentypen als Strings enthalten. Zellen können unterschiedliche Einstellungen für Schriftart, -größe und -farbe aufweisen und unterschiedliche Breiten und Höhen haben. Benachbarte Zellen lassen sich verschmelzen und es ist möglich, Bilder und Diagramme einzubetten.
2. Sie übergeben das von `open()` zurückgegebene `File`-Objekt.
3. `File`-Objekte müssen im binären Lesemodus ('`rb`', für Reader-Objekte) bzw. im binären Schreibmodus ('`wb`', für Writer-Objekte) geöffnet werden.
4. Die Methode `writerow()`
5. Das Argument `delimiter` ändert den String, der als Trennzeichen für die Zellen in einer Zeile verwendet wird, das Argument `lineterminator` das Trennzeichen zwischen den Zeilen.
6. `json.loads()`
7. `json.dumps()`

## Kapitel 15

1. Ein Referenzzeitpunkt, an dem sich viele Datums- und Uhrzeitprogramme orientieren. Dieser Zeitpunkt ist der 1. Januar 1970 UTC.
2. `time.time()`
3. `time.sleep(5)`
4. Sie gibt den Integerwert zurück, der dem übergebenen Argument am nächsten liegt, beispielsweise 2 für `rund(2.4)`.
5. Ein `datetime`-Objekt steht für einen bestimmten Zeitpunkt, ein `timedelta`-Objekt für eine Zeitdauer.
6. `threadObj = threading.Thread(target=spam)`
7. `threadObj.start()`

8. Sorgen Sie dafür, dass der Code in einem Thread nicht in denselben Variablen liest und schreibt wie Code in einem anderen Thread.
9. `subprocess.Popen('c:\\Windows\\System32\\calc.exe')`

## Kapitel 16

1. SMTP bzw. IMAP
2. `smtplib.SMTP()`, `smtpObj.ehlo()`, `smtpObj.starttls()` und `smtpObj.login()`
3. `imapclient.IMAPClient()` und `imapObj.login()`
4. Eine Liste von Strings für IMAP-Schlüsselwörter wie '`BEFORE <date>`', '`FROM <string>`' oder '`SEEN`'
5. Weisen Sie der Variablen `imaplib._MAXLINE` einen großen Integerwert zu, z. B. `10000000`.
6. Das Modul `pymail` liest heruntergeladene E-Mails.
7. Sie brauchen die SID-Nummer des Twilio-Kontos, die Nummer des Authentifizierungstokens und Ihre Twilio-Telefonnummer.

## Kapitel 17

1. Ein RGBA-Wert ist ein Tupel aus vier Integern jeweils aus dem Bereich von 0 bis 255. Sie stehen für den Anteil von Rot, Grün und Blau und die Transparenz (Alpha) der Farbe.
2. Der Aufruf der Funktion `ImageColor.getcolor('CornflowerBlue', 'RGBA')` gibt `(100, 149, 237, 255)` zurück, den RGBA-Wert dieser Farbe.
3. Ein Rechtecktupel ist ein Tupelwert aus vier Integern, die für die x-Koordinate der linken Kante, die y-Koordinate der oberen Kante, die Breite und die Höhe stehen.
4. `Image.open('zophie.png')`
5. `imageObj.size` ist ein Tupel aus zwei Integern für die Breite und die Höhe.
6. `imageObj.crop((0, 50, 50, 50))`. Beachten Sie, dass Sie an `crop()` ein Rechtecktupel übergeben und keine vier getrennten Integerargumente.
7. Rufen Sie die Methode `imageObj.save('new_filename.png')` des `Image`-Objekts auf.
8. Das Modul `ImageDraw` enthält Code, um auf Bildern zu zeichnen.
9. `ImageDraw`-Objekte verfügen über Methoden zum Zeichnen von Formen wie `point()`, `line()` und `rectangle()`. Sie werden zurückgegeben, wenn Sie ein `Image`-Objekt an die Funktion `ImageDraw.Draw()` übergeben.

## Kapitel 18

1. Verschieben Sie den Mauszeiger in die obere linke Ecke des Bildschirms, also zu den Koordinaten (0, 0).
2. pyautogui.size() gibt ein Tupel mit zwei Integern für die Breite und Höhe des Bildschirms zurück.
3. pyautogui.position() gibt ein Tupel mit zwei Integern für die x- und y-Koordinate des Mauszeigers zurück.
4. Die Funktion moveTo() verschiebt den Mauszeiger auf die angegebenen absoluten Koordinaten auf dem Bildschirm, während moveRel() ihn relativ zu seiner jetzigen Position verschiebt.
5. pyautogui.dragTo() und pyautogui.dragRel()
6. pyautogui.typewrite('Hello world!')
7. Übergeben Sie entweder eine Liste der Tastenstrings an pyautogui.typewrite() (z. B. 'left') oder einen einzelnen Tastenstring an pyautogui.press().
8. pyautogui.screenshot('screenshot.png')
9. pyautogui.PAUSE = 2

# Stichwortverzeichnis

## Symbol

<code>^</code>	183
Beginn eines Strings	180
Negative Zeichenklassen	180
<code>-</code>	23
<code>-</code>	17
<code>:</code>	44, 51, 61, 92
<code>!=</code>	37
<code>?</code>	174, 183
<code>.</code>	181
<code>..</code>	197
<code>:*</code>	181
<code>:</code>	140
<code>"</code>	140
<code>'''</code>	142, 185
<code>()</code>	108, 171
<code>[]</code>	90, 183
<code>{}</code>	119, 183
Gierig/nicht gierig	177
Wiederholungen	176
<code>*</code>	17, 94, 99, 182
Null oder mehr Übereinstimmungen	174
<code>**</code>	17
<code>/</code>	196
Division	17, 99
<code>//</code>	17
<code>\</code>	140, 169, 182, 196
Zeilenfortsetzungszeichen	105
<code>#</code>	142
<code>+</code>	94, 175, 183
Addition	17, 20, 99
<code>&lt;</code>	37
<code>&lt;=</code>	37
<code>=</code>	21, 39
<code>==</code>	37, 39
<code>&gt;</code>	37
<code>&gt;=</code>	37
<code> </code>	173, 185
<code>\$</code>	180, 183
<code>%a</code>	396
<code>%A</code>	396
<code>\b</code>	488
<code>%b</code>	396

<code>%B</code>	396
<code>\d</code>	178
<code>%d</code>	396
<code>\D</code>	178
<code>%H</code>	396
<code>%I</code>	396
<code>%j</code>	396
<code>%m</code>	396
<code>%M</code>	396
<code>%p</code>	396
<code>\s</code>	179
<code>\S</code>	179
<code>%S</code>	396
<code>\w</code>	178
<code>%w</code>	396
<code>\W</code>	178
<code>%y</code>	396
<code>%Y</code>	396

## A

Abgerundeter Quotient	17
Absätze	353
Absolute Pfade	198
abspath()	201
Absturz	17
add_heading()	360
Addition	17, 20, 99
Additives Farbmodell	451
addPage()	342
add_paragraph()	358
add_picture()	361
add_run()	358
Aktuelles Arbeitsverzeichnis	197
Algebraische Schachnotation	127
ALL	427
all_caps	356
Alpha	450
and	39
ANSWERED	427
Anwendungsspezifische Passwörter	421
API (Anwendungsprogrammierschnittstelle)	377
append()	101

- Arbeitsmappen 305  
 Arbeitsblätter erstellen 318  
 Arbeitsblätter löschen 319  
 Öffnen 305  
 Speichern 317  
 args 401  
 Argumente 27, 71  
 Prozesse 407  
 Schlüsselwortargumente 74  
 Threads 400  
 AT&T-Mail 419, 424  
 Attribute 277, 284  
 Auflösung 484  
 Aufrufstack 248  
 Aufzählungspunkte zu Wiki-Markup hinzufügen 156  
 Ausdrücke  
   Bedingungen 42  
   Interaktive Shell 16  
 Ausnahmen  
   Auslösen 246  
   Behandeln 82  
   Traceback 248  
   Zusicherungen 249  
 Auswahllisten 508  
 Auswertungsreihenfolge von Operatoren 17
- B**
- back() 299  
 Backslash 140, 170, 183, 196  
 BarChart() 332  
 basename() 202  
 BCC 427  
 BeautifulSoup 281  
 Bedingungen 42  
 Beendigungscodes 407  
 Befehlszeilenargumente 269  
 BEFORE 427  
 Bilder  
   Attribute 455  
   Beschneiden 456  
   Drehen 461  
   Ellipsen 472  
   Erkennen 495  
   Farbwerte 450  
   Größe ändern 460  
   ImageDraw 471  
   Koordinaten 452  
   Kopieren und einfügen 457  
   Linien 471  
   Logo hinzufügen 464
- Mehrere Bilder herunterladen 402  
 Öffnen mit Pillow 454  
 Pixelbearbeitung 463  
 Polygone 472  
 Punkte 471  
 Rechtecke 472  
 Rechtektupel 452  
 RGBA 450  
 Spiegeln 461  
 Text 473  
 Transparente Pixel 460  
 Zeichnen 473  
 Bildschirm  
   Auflösung 484  
   Koordinaten 484  
 Binärdateien 204, 209  
 Binäre Operatoren 39  
   Binär- und Vergleichsoperatoren 41  
   Flusssteuerung 36  
   in 97  
   not in 97  
   Truthy und falsey 60  
 Bitweises Or 186  
 Blockieren der Ausführung 388  
 BODY 427  
 bold 356  
 break  
   for-Schleifen 62  
   Übersicht 56  
 Browser öffnen 268  
 Browertabs 287  
 bs4  
   Attribut abrufen 284  
   Element mit select() finden 282  
   Objekt aus HTML erstellen 281  
   Übersicht 281
- C**
- CamelCase 24  
 CC 427  
 Cell 306  
 center() 150, 496  
 chdir() 197  
 Chrome 278  
 clear() 296  
 click() 297, 489, 501, 503  
 CMYK 451  
 Codeblöcke 42  
 Collatz-Folge 87  
 column\_index\_from\_string() 308  
 Comcast 419, 424  
 commentAfterDelay() 501  
 compile() 170, 171, 185

- continue  
for-Schleifen 62  
Übersicht 57
- Coordinated Universal Time (UTC) 386
- copy() 113, 152, 224, 457
- copytree() 225
- Countdown 411
- cProfile.run() 387
- create\_sheet() 318
- CRITICAL 256
- cron 408
- CSS  
Selektoren 282  
Selenium 296
- CSV-Dateien  
Daten lesen 370  
Definition 367  
Format 368  
In einer Schleife durchlaufen 373  
Kopfzeilen entfernen 373  
Lesen 374  
Reader-Objekte 369  
Schreiben 375  
Trennzeichen 372  
Übersicht 368  
Writer-Objekte 371  
Zeilenendezeichen 372
- D**
- Dateieditor 24
- Dateien  
Absolute und relative Pfade 198  
Aktuelles Arbeitsverzeichnis 197  
Dateien und Ordner verschieben 225  
Dateinamen 195  
Entpacken 232  
Erweiterungen 195  
Komprimierte Dateien 230, 238  
Lesen 206  
Mehrfach-Zwischenablage 216  
Öffnen 205  
os.path 200  
Pfade 195  
pformat() 210  
Schreiben 207  
send2trash 228  
shelve 208  
shutil 224  
Text- und Binärdateien 204  
Übersicht 195
- Umbenennen 234  
Verzeichisse erstellen 199  
Verzeichnisbäume durchlaufen 228  
ZIP-Dateien erstellen 233  
ZIP-Dateien lesen 231
- Datenstrukturen  
Algebraische Schachnotation 127  
Tic-Tac-Toe 128
- Datentypen  
Boolesche Werte 36  
Definition 19  
Dictionarys 119  
Fließkommazahlen 19  
Integer 19  
list() 110  
Listen 89  
None 73  
Strings 19  
Tupel 108  
tuple() 110  
Veränderbare und nicht veränderbare 106
- datetime  
fromtimestamp() 393  
Kalenderarithmetik 394  
now() 392  
Objekte in Strings umwandeln 396  
Programme bis zu einem bestimmten Zeitpunkt anhalten 395  
Strings in Objekte umwandeln 397  
timedelta 394  
total\_seconds() 394  
Übersicht 392, 398
- debug() 253  
DEBUG 254
- Debugging  
Ausnahmen 246  
Haltepunkte 262  
IDLE 257  
logging 252  
print() 254  
Programm schrittweise durchlaufen 257  
Protokolliergrad 255  
Protokollierung 257  
Protokollierung deaktivieren 256  
Traceback 248  
Zusicherungen 249
- decode() 433  
deepcopy() 113  
def 70  
Parameter 71  
del 94

- DELETED 427  
delete\_messages() 433  
delimiter 372  
Diagramme 330  
Dictionarys  
    copy() 113  
    deepcopy() 113  
    get() 123  
    in 123  
    items() 122  
    keys() 122  
    Listen 120  
    not in 123  
    setdefault() 124  
    Übersicht 119  
    values() 122  
    Veschachtelt 133  
dirname() 202  
disable() 256  
Division 17, 99  
Document-Objekte 350  
Dollarzeichen 180, 183  
Doppelkreuz 142  
Doppelpunkt 44, 51, 61, 92  
Doppel Anführungszeichen 140  
doubleClick() 490, 501  
double\_strike 356  
DRAFT 427  
dragRel() 490, 491, 501  
dragTo() 490, 501  
Dreifache Anführungszeichen 141, 185  
Drittanbietermodule 513  
dumps() 378  
Duplizierter Code 70  
duration 485
- E**  
Eckige Klammern 89  
ehlo() 420, 438  
Einfache Anführungszeichen 140  
Einrückung 105  
Einzelthreadprogramme 399  
elif 45  
ellipse() 472  
else 44  
E-Mail-Adressen 186  
    Reguläre Ausdrücke 188  
    Übereinstimmungen in der Zwischenablage finden 189  
Übereinstimmungen zu einem String zusammensetzen 190  
Übersicht 187
- E-Mails 420  
    Abrufen 423  
    Adressen aus Rohnachrichten entnehmen 431  
    Als gelesen kennzeichnen 430  
    Anmelden am Server 421, 425  
    gmail\_search() 430  
    IMAP 423  
    Löschen 433  
    Nachrichten abrufen 430  
    Ordner abrufen 425  
    Rohnachrichten 431  
    Senden 418, 422  
    SMTP 418  
    Suchen 425  
    TLS-Verschlüsselung 421  
    Trennen vom Server 422, 434  
    Verbindung mit dem SMTP-Server 419  
emboss 356  
Endlosschleifen 56, 58, 488  
endswith() 148  
Entpacken 232  
Entschlüsselung 340  
Epochenzzeitstempel 386, 392, 398  
ERROR 256  
Erweiterte Zuweisungsoperatoren 99  
Excel-Arbeitsblätter  
    Aktualisieren 319  
    Arbeitsblätter erstellen 318  
    Arbeitsblätter löschen 319  
    Arbeitsblattnamen abrufen 306  
    Arbeitsmappen 305  
    Arbeitsmappen speichern 317  
    Dateien lesen 311  
    Daten lesen 312  
    Datenstruktur füllen 313  
    Diagramme 330  
    Dokumente erstellen 317  
    Dokumente öffnen 305  
    Einrichten 320  
    Ergebnisse in Datei schreiben 315  
    Fixieren 329  
    Formeln 325  
    openpyxl 304  
    Schriftarten 323  
    Spaltenbreiten 327  
    Spaltenbuchstaben und -nummern umwandeln 308  
    Übersicht 304  
    Unterstützung 303  
    Werte in Zellen schreiben 319  
    Zeilenhöhe 327

- Zeilen und Spalten abrufen 309  
Zellen verbinden und aufteilen 328  
Zellenwerte abrufen 306  
Exception-Objekte 247  
exists() 204  
expand 462  
Exponent 17  
expunge() 433  
extract() 232  
extractall() 232
- F**
- FailSafeException 507  
Falsey 60  
Farbwerte  
    CMYK/RGB 451  
    RGBA 450  
Fehler 17  
fetch() 429, 430  
File-Objekte 206  
findall() 177  
find\_element\_by\_\* 295  
find\_elements\_by\_\* 295  
Firefox 279  
FLAGGED 428  
Fließkommazahlen  
    Runden 388  
    Übersicht 19  
float() 29  
Flussteuerung  
    Bedingungen 42  
    Binäre Operatoren 39  
    Binär- und Vergleichsoperatoren 41  
    Boolesche Werte 36  
    break 56  
    Codeblöcke 42  
    continue 57  
    elif 45  
    else 44  
    if 43  
    Übersicht 35  
    Vergleichsoperatoren 37  
    while-Schleifen 51  
Font-Objekte 324  
format 455  
format\_description 455  
formData 506  
Formeln (Excel) 325  
Formulare automatisch ausfüllen 502  
    Ablauf ermitteln 503  
    Auswahllisten 508  
    Daten eingeben 506
- Formular absenden 509  
Koordinaten 503  
Optionsschalter 508  
Übersicht 501  
for-Schleifen  
    Dictionary-Elemente 122  
    Listen 96  
    Übersicht 60  
forward() 299  
Fragezeichen 174, 183  
FROM 427  
fromtimestamp() 393, 398  
Funktionen  
    Argumente 27, 71  
    Aufrufen 27  
    Ausnahmebehandlung 82  
    Blackbox 82  
    def 71  
    Integriert 65  
    None 73  
    Parameter 71  
    Rückgabewerte 72  
    Schlüsselwortargumente 74  
    Übersicht 69
- G**
- Geschweifte Klammern 119, 183  
Gierig/nicht gierig 177  
Wiederholungen 176  
get()  
    requests 271  
    Übersicht 123  
get\_active\_sheet() 306  
get\_addresses() 432  
get\_attribute() 296  
getcolor() 450, 456  
get\_column\_letter() 308  
getcwd() 197  
get\_highest\_column() 308, 436  
get\_highest\_row() 308  
get\_payload() 433  
getpixel() 463, 494  
get\_sheet\_by\_name() 306  
get\_sheet\_names() 306  
getsize() 203  
get\_subject() 432  
getText() 352  
Gierige Suche 181  
GIF 455  
Gleichheit 37, 39  
Globaler Gültigkeitsbereich 79  
Gmail 419, 421, 424

- gmail\_search() 430  
 Google Maps 268  
 Größer als 37  
 Größer oder gleich 37  
 Groß- und Kleinschreibung 24, 183  
 group() 170, 171  
 Gruppen (reguläre Ausdrücke)  
   Eine oder mehr 175  
   Genaue Anzahl von Wieder-  
     holungen 176  
 Gierig 177  
 Klammern 171  
 Nicht gierig 177  
 Null oder mehr 174  
 Optional 174  
 Pipe 173
- GUI-Automatisierung  
 Bilderkennung 495  
 Drücken und loslassen 500  
 Mausklick 489  
 Mausposition bestimmen 486  
 Maus steuern 484, 489  
 Maus ziehen 490  
 Programm beenden 483  
 pyautogui 482  
 Screenshots 493  
 Scrollen 492  
 String von der Tastatur senden 497  
 Tastatur steuern 497  
 Tastenkombinationen 500  
 Tastennamen 498  
 Übersicht 481  
 Vom Programm abmelden 483
- Gültigkeitsbereich  
 Global 79  
 Lokal 75
- H**
- Haltepunkte 262  
 Herunterladen  
   Dateien aus dem Web 274  
   Webseiten 271  
   XKCD-Comics 288, 403  
 hotkey() 500, 502  
 Hotmail.com 419, 424  
 HTML  
   Elemente finden 280  
   Entwicklertools 278  
   Lernressourcen 276  
   Quelltext 277  
   Übersicht 276
- I**
- id 277  
 IDLE  
   Ausdrücke 16  
   Debugging 257  
   Haltepunkte 262  
   Programme erstellen 24  
   Programm schrittweise durch-  
     laufen 257  
   Skripts außerhalb von IDLE aus-  
     führen 153
- if  
   Übersicht 43  
   while-Schleifen 52
- im 493  
 Image 454  
 imageDraw 471  
 ImageFont 473  
 IMAP  
   Anmelden am Server 425  
   Definition 423  
   Nachrichten abrufen 430  
   Nachrichten löschen 433  
   Nachrichten suchen 425  
   Ordner 425  
   Trennen vom Server 434
- imapclient 423  
 IMAPClient 424  
 Import von Modulen  
   pyautogui 486  
   Übersicht 65
- imprint 356  
 in  
   Dictionarys 123  
   Listen 97  
   Strings 144
- index() 100  
 IndexError 120  
 Indizes  
   Negativ 92  
   Strings 142  
   Werte abrufen 90  
   Werte ändern 93  
   Werte aus Listen entfernen 94
- INFO 255  
 input()  
   Sensible Informationen 422  
   Übersicht 27
- insert() 101  
 Installation  
   Drittanbietermodule 513  
   openpyxl 304  
   pyautogui 482  
   Selenium 294

- int 19  
int() 29  
Integer  
  Strings umwandeln 31  
  Übersicht 19  
Integerdivision 17  
Integrierte Funktionen 65  
Internet Explorer 278  
isabs() 200  
isalnum() 146  
isalpha() 146  
isdecimal() 146  
isdir() 203  
is\_displayed() 296  
is\_enabled() 296  
isfile() 203  
islower() 144  
is\_selected() 297  
isspace() 146  
istitle() 146  
isupper() 144  
italic 356  
items() 122  
iter\_content() 274
- J**  
join() 196, 200, 405  
Jokerzeichen 181  
JPEG 455  
JSON-Dateien  
  APIs 376  
  Definition 367  
  Format 376  
  Lesen 378  
  Schreiben 378  
  Wetterdatenprojekt 379
- K**  
Kalenderarithmetik 394  
Karten  
  Befehlszeilenargumente 269  
  Browser starten 270  
  Übersicht 269  
  URL finden 269  
  Zwischenablage 270  
KeyboardInterrupt 391, 486, 487  
keyDown() 500, 502  
keys() 122  
keyUp() 500, 502  
Klammern 109, 171  
Klangdateien 412  
Kleiner als 37  
Kleiner oder gleich 37  
Kommentare  
  Mehrzeilig 142  
  Übersicht 26  
Komprimierte Dateien  
  Entpacken 232  
  Erstellen 233  
  Lesen 231  
  Ordner sichern 238  
  Übersicht 230  
Koordinaten  
  Bilder 452  
  Bildschirm 483
- L**  
LARGER 428  
launchd 408  
Leere Strings 19  
len()  
  Anzahl der Werte in einer Liste  
  finden 93  
  Übersicht 28  
LibreOffice 303, 350  
line() 471  
LineChart() 332  
lineterminator 372  
Linux  
  cron 408  
  Dateien mit Standardanwendungen  
  öffnen 409  
Drittanbietermodule 513  
Pfadtrennzeichen 196  
pip 513  
Prozesse von Python aus  
  starten 406  
Python-Programme ausführen 519  
Unix-Philosophie 410  
Vom Automatisierungsprogramm  
  abmelden 483  
list() 369, 496  
listdir() 203  
Listen  
  Anzahl der Werte mit len()  
  finden 93  
  append() 101  
  copy() 113  
  deepcopy() 113  
  Dictionarys 120  
  Erweiterte Zuweisungs-  
    operatoren 99  
  for-Schleifen 96  
  in 97

index() 100  
 insert() 101  
 list() 110  
 Magic 8 Ball 104  
 Mehrfachzuweisung 98  
 Negative Indizes 92  
 not in 97  
 remove() 102  
 sort() 103  
 Teillisten abrufen 92  
 Übersicht 89  
 Variablen speichern 94  
 Veränderbare und unveränderbare Datentypen 106  
 Verkettung 94  
 Verschachtelt 133  
 Werte ändern 93  
 Werte anhand des Index abrufen 90  
 Werte entfernen 94  
 Wiederholung 94  
 list\_folders() 425  
 ljust() 150  
 loads() 378, 381  
 load\_workbook() 305  
 locateAllOnScreen() 496  
 locateOnScreen() 496  
 location 297  
 logging 252  
 login() 421, 425, 438  
 Logo hinzufügen 464  
 logout() 434  
 LogRecord 252  
 Lokaler Gültigkeitsbereich 76  
 Löschen  
     Endgültig 227  
     send2trash 228  
 lower() 144  
 lstrip() 152

**M**

Mac OS X  
     Dateien mit Standardanwendungen öffnen 409  
     Drittanbietermodule 513  
     launchd 408  
     Pfadtrennzeichen 196  
     pip 513  
     Prozesse von Python aus starten 406, 410  
     Python-Programme ausführen 519  
     Unix-Philosophie 410  
     Vom Automatisierungsprogramm abmelden 483

Magic 8 Ball 104  
 makedirs() 199, 467  
 Maskierungszeichen 140  
 Match-Objekte 170  
 Maus  
     Endlosschleifen 488  
     KeyboardInterrupt 487  
     Koordinaten abrufen 488  
     Mausklick 489  
     Maus ziehen 490  
     Pixelfarben 494  
     Position bestimmen 486  
     pyautogui 487  
     Scrollen 492  
     Steuern 484, 489  
     Übersicht 484  
 Mehrfachzuweisung 98  
 Mehrfach-Zwischenablage 216  
     Inhalt der Zwischenablage speichern 218  
     Schlüsselwörter auflisten 219  
     Shelf-Datei 218  
     Übersicht 216  
 Mehrzeilige Kommentare 142  
 Mehrzeilige Strings 141  
 mergePage() 344  
 Message-Objekte 441  
 Methoden  
     append() 101  
     Aufrufe verketten 461  
     center() 150  
     copy() 152  
     Definition 100  
     endswith() 148  
     get() 123  
     index() 100  
     insert() 101  
     isalnum() 146  
     isalpha() 146  
     isdecimal() 146  
     islower() 144  
     isspace() 146  
     istitle() 146  
     isupper() 144  
     items() 122  
     join() 149  
     keys() 122  
     ljust() 150  
     lower() 144  
     lstrip() 152  
     paste() 152  
     remove() 102

- rjust() 150  
rstrip() 152  
setdefault() 124  
sort() 103  
split() 149  
startswith() 148  
strip() 152  
upper() 144  
values() 122  
middleClick() 490, 501  
**M**odule  
    Drittanbietermodule 513  
    Importieren 65  
    Modulus 17  
Modulus 99  
mouseDown() 490, 501  
mouse.position() 487  
mouseUp() 502  
move() 225  
moveRel() 485, 486, 490, 501  
moveTo() 485, 490, 501  
Multiplikation 17, 99  
**M**ultithreading  
    Argumente an Threads  
        übergeben 401  
    Auf Threads warten 405  
    downloadXkcd() 403  
    join() 405  
    Mehrere Bilder herunterladen 402  
    Probleme der Nebenläufigkeit 402  
    start() 400, 401  
    Thread() 399  
    Threads erstellen und starten 404  
    Übersicht 399
- N**ameError 94  
namelist() 231  
Nebenläufigkeit 402  
Negative Indizes 92  
Negative Zeichenklassen 180  
newline 371  
Nicht gierig 182  
None 73  
not 40  
NOT 428  
not in  
    Dictionarys 123  
    Listen 97  
    Strings 144  
now() 392, 398
- O**ON 426  
open 408, 409  
open() 205, 454  
OpenOffice 303, 350  
openpyxl 304  
Operatoren  
    Binär 39  
    Binär- und Vergleichsoperatoren 41  
    Definition 17  
    Erweiterte Zuweisungs-  
        operatoren 99  
    Mathematik 17  
    Rangfolge 17  
    Vergleichsoperatoren 37  
Optionsschalter 508  
or 40, 428  
Ordner  
    Absolute und relative Pfade 198  
    Aktuelles Arbeitsverzeichnis 197  
    Backslash und Schrägstrich 196  
    Definition 196  
    Endgültig löschen 227  
    Erstellen 199  
    Kopieren 224  
    send2trash 228  
    Sichern 238  
    Trennzeichen 196  
    Umbenennen 225  
    Verschieben 225  
    Verzeichnisbäume durchlaufen 228  
os.path  
    Absolute Pfade 200  
    Dateigrößen 202  
    Gültigkeit von Pfaden 203  
    Ordnerinhalte 202  
    Relative Pfade 200  
    Übersicht 200  
outline 356  
Outlook.com 419, 424
- P**age-Objekte 339  
Paragraph-Objekte 350  
Parameter 71  
Parse 281  
Passwörter  
    Anwendungsspezifisch 421  
    Befehlszeilenargumente 154  
    Datenstrukturen 154  
    Kopieren 155  
    Passwortsafe 153  
    Übersicht 153

- paste() 152, 457, 458  
PAUSE 483, 506  
PDF-Dateien  
    Entschlüsseln 340  
    Ergebnisse speichern 349  
    Erstellen 340  
    Format 337  
    Öffnen 347  
    Seiten aus mehreren Dateien kombinieren 346  
    Seiten drehen 342  
    Seiten finden 346  
    Seiten hinzufügen 348  
    Seiten kopieren 341  
    Seiten überlagern 344  
    Text entnehmen 338  
    Verschlüsseln 345  
PdfFileReader 339  
PdfFileWriter 340  
Pfade  
    Absolute und relative Pfade 198  
    Aktuelles Arbeitsverzeichnis 197  
    Backslash und Schrägstrich 196  
    os.path 200  
    Trennzeichen 196  
    Übersicht 195  
pformat()  
    Übersicht 125  
    Variablen in Textdateien speichern 210  
Pillow  
    Beispielprogramm 472  
    Bildattribute 455  
    Bilder beschneiden 456  
    Bilder drehen 461  
    Bilder kopieren und einfügen 457  
    Bilder öffnen 454  
    Ellipsen 472  
    Größe von Bildern ändern 460  
    ImageDraw 471  
    Linien 471  
    Modul 450  
    Pixelbearbeitung 463  
    Polygone 472  
    Punkte 471  
    Rechtecke 472  
    Text 473  
    Transparente Pixel 460  
pip 513  
Pipe 173, 186  
Pixel 450, 463  
pixelMatchesColor() 494, 507  
Pluszeichen 175, 183  
PNG 455  
point() 471  
poll() 407  
polygon() 472  
Popen() 405  
    Befehlszeilenargumente übergeben 407  
    Dateien mit Standardanwendungen öffnen 409  
position() 486, 488  
 pprint() 125  
press() 500, 502, 509  
print()  
    Protokollierung 252  
    Übersicht 26  
Programmausführung  
    Anhalten 395  
    Definition 35  
    Programm mit sys.exit()  
        beenden 66  
    Übersicht 43  
Programme  
    Befehlszeilenargumente übergeben 407  
    Countdown 411  
    Dateien mit Standardanwendungen öffnen 409  
    Linux 519  
    Mac OS X 519  
    poll() 407  
    Python-Skripte 408  
    Shebang-Zeile 517  
    sleep() 408  
    wait() 407  
    Websites öffnen 408  
    Windows 518  
    Zeitplan 408  
Programmierung  
    Ausnahmebehandlung 82  
    Blackbox 82  
    Codeblöcke 42  
    Duplizierter Code 70  
    Einrückung 105  
    Globaler Gültigkeitsbereich 79  
    Kommentare 26  
    Lokaler Gültigkeitsbereich 76  
    Programmausführung 43  
    Programm mit sys.exit()  
        beenden 66  
Projekte  
    Aktuelle Wetterdaten abrufen 379  
    Alle XKCD-Comics herunterladen 288

- Amerikanische Datumsangaben in europäische ändern 233  
Arbeitsblätter aktualisieren 319  
Aufzählungspunkte zu einem Wiki- Markup hinzufügen 156  
Ausgewählte Seiten aus mehreren PDFs kombinieren 346  
Daten in einem Arbeitsblatt lesen 311  
Einfaches Countdown-Programm 411  
E-Mails über ausstehende Mitgliedsbeiträge senden 435  
Formulare automatisch ausfüllen 502  
Google-Suche 285  
Kopfzeilen einer CSV-Datei entfernen 373  
Logo hinzufügen 464  
mapIt.py mit webbrowser 268  
Mehrfach-Zwischenablage 216  
Modul 443  
mouseNow erweitern 495  
Multithread-Version des XKCD-Downloadprogramms 402  
Ordner in einer ZIP-Datei sichern 238  
Passwortsasfe 153  
Superstoppuhr 389  
Telefonnummern und E-Mail-Adressen extrahieren 186  
Wo ist mein Mauszeiger? 486  
Zufällige Testfragen generieren 211
- Protokollierung  
Deaktivieren 256  
In Datei 257  
print() 254  
Protokolliergrade 255
- Prozesse  
Befehlszeilenargumente übergeben 407  
Countdown 411  
Dateien mit Standardanwendungen öffnen 409  
Definition 405  
poll() 407  
Popen() 405  
wait() 407  
Websites öffnen 408
- Punkt 181  
Jokerzeichen 181  
Pfade 197  
Punkt-Stern 181
- putpixel() 463  
pyautogui  
Bilderkennung 495  
Dokumentation 482  
Drücken und loslassen 500  
Failsafe 483  
Formulare automatisch ausfüllen 502  
Funktionen 501  
Funktionsaufrufe anhalten 483  
Importieren 483  
Installation 482  
Mausklick 489  
Maus steuern 489  
Maus ziehen 490  
Screenshots 493  
Scrollen 492  
String von der Tastatur senden 497  
Tastatur steuern 497  
Tastenkombinationen 500  
Tastennamen 498  
pyautogui.click() 489, 503  
pyautogui.doubleClick() 490  
pyautogui.dragTo() 490  
pyautogui.FailSafeException 483  
pyautogui.hotkey() 500  
pyautogui.keyDown() 500  
pyautogui.keyUp() 500  
pyautogui.middleClick() 490  
pyautogui.mouseDown() 490  
pyautogui.moveRel() 485, 486  
pyautogui.moveTo() 485  
pyautogui.PAUSE 483  
pyautogui.position() 488  
pyautogui.press() 509  
pyautogui.rightClick() 490  
pyautogui.screenshot() 493  
pyautogui.size() 484  
pyautogui.typewrite() 497, 498  
py.exe 518  
pyobjc 515  
PyPDF2  
Format 337  
PDFs entschlüsseln 340  
PDFs erstellen 340  
PDFs verschlüsseln 345  
Seiten aus mehreren PDFs kombinieren 346  
Seiten drehen 342  
Seiten kopieren 341  
Seiten überlagern 344  
Text aus PDFs entnehmen 338  
pyperclip 152

- Python  
 Beispielprogramm 24  
 Datentypen 19  
`python-docx` 350  
`pymail` 423, 431  
`PyzMessage` 431
- Q**  
`quit()` 299, 422, 438
- R**  
`raise` 246  
`raise_for_status()` 273  
`range()` 63  
`read()` 206  
 Reader-Objekte 369  
`readlines()` 206  
 Rechtecktupel 452  
`rectangle()` 472  
 Reference-Objekte 331  
`refresh()` 299  
 Regex-Objekte  
     Erstellen 169  
     Übereinstimmungen 170  
 Reguläre Ausdrücke  
     Argumente für `compile()` 185  
     Beginn von Strings 180  
     Definition 165  
     Eine oder mehr 175  
     Ende von Strings 180  
     `findall()` 177  
     Gierige Suche 177  
     Groß- und Kleinschreibung 183  
     HTML 279  
     Jokerzeichen 181  
     Kalenderdaten 234  
     Klammern 171  
     Mehrere Zeilen 185  
     Muster 168  
     Nicht gierige Suche 177  
     Null oder mehr 174  
     Optionale Übereinstimmungen 174  
     Pipe 173  
     Regex-Objekte 169  
     Strings ersetzen 184  
     Symbolverzeichnis 183  
     Telefonnummern und E-Mail-  
     Adressen 186  
     Text ohne reguläre Ausdrücke  
     finden 166  
     Übereinstimmungen 170  
     Wiederholungen 176  
     Zeichenklassen 178
- Reine Textdateien 204  
 Relative Pfade 198  
`relpath()` 200, 201  
`remove()` 102  
`remove_sheet()` 318  
`requests`  
     Dateien herunterladen 274  
     Seiten herunterladen 271  
 Response-Objekte 272  
 Rest 17, 99  
`reverse` 103  
`RGBA` 450  
 RGB-Farbmodell 451  
`rightClick()` 490, 501  
`rjust()` 150, 488  
`rmdir()` 227  
`rmtree()` 227  
 Rohstrings 141, 170  
`rotateClockwise()` 342  
`rotateCounterClockwise()` 342  
`rstrip()` 152  
`rtl` 356  
 Rückgabewerte 72  
 Runden 388  
 Run-Objekte 354, 356
- S**  
 Safari 279  
`save()` 454  
 Schleifen  
     `break` 56  
     `continue` 57  
     Daten aus CSV-Dateien lesen 370  
     for-Schleifen 61  
     Listen 96  
     `range()` 63  
     while-Schleifen 51  
 Schlüssel 119  
 Schlüsselwortargumente 74  
 Schrägstrich 196  
 Schriftarten 323  
 Schrittweite 64  
`screenshot()` 493, 502  
 Screenshots  
     Analysieren 494  
     Erstellen 493  
`scroll()` 492, 493, 502  
 Scrollen 492  
`search()` 170  
`see` 409  
 SEEN 427  
 Seitenumbrüche 360

- select() 282  
select\_folder() 426  
Selektoren 282, 296  
Selenium  
    Auf Schaltflächen klicken 299  
    Elemente finden 295  
    Firefox 295  
    Formulare absenden 298  
    Installation 294  
    Links folgen 297  
    Sondertasten 298  
send2trash 228  
send\_keys() 298  
sendmail() 422, 439  
Sequenzen 97  
Sequenznummern 431  
setdefault() 124  
shadow 356  
Shebang-Zeile 517  
shelve 208  
shutil  
    Dateien und Ordner kopieren 224  
    Dateien und Ordner löschen 227  
    Dateien und Ordner umbenennen 225  
    Dateien und Ordner verschieben 225  
SID 442  
SINCE 427  
size() 484  
Skripts  
    Ausführen 408  
    Außerhalb von IDLE ausführen 153  
sleep() 387, 395, 398, 408  
Slices  
    Strings 142  
    Teillisten 92  
small\_caps 356  
SMALLER 428  
SMS  
    Nachrichten senden 440  
    Twilio 439  
SMTP 420  
    Anmelden am Server 421  
    Definition 418  
    Nachrichten senden 422  
    SMTP-Objekte 419  
    TLS-Verschlüsselung 421  
    Trennen vom Server 422  
    Verbindung zum Server 419  
sort() 103
- Spalten (Excel)  
    Breite 327  
    Cell-Objekte abrufen 309  
split() 149, 201, 368  
Standardbibliothek 65  
start() 400, 401, 404  
startswith() 148  
starttls() 421  
Sternchen 181, 183  
Jokerzeichen 181  
Null oder mehr Übereinstimmungen 174  
Stoppuhr 389  
    Übersicht 389  
Vorbereiten 390  
Zeit messen 390  
str() 29, 488  
strftime() 396, 398  
strike 356  
String-ID 442  
Strings  
    center() 150  
    datetime-Objekte 396, 397  
    Doppelte Anführungszeichen 140  
    endswith() 148  
    Ersetzen 184  
    in 143  
    Indizes 143  
    In Integer umwandeln 31  
    isalnum() 146  
    isalpha() 146  
    isdecimal() 146  
    islower() 144  
    isspace() 146  
    istitle() 146  
    isupper() 144  
    join() 149  
    Kopieren und einfügen 152  
    Literale 140  
    ljust() 150  
    lower() 144  
    lstrip() 152  
    Maskierung 140  
    Mehrzeilig 141  
    not in 144  
    PDF-Text extrahieren 338  
    rjust() 150  
    Rohstrings 141  
    rstrip() 152  
    Slices 143  
    split() 149  
    startswith() 148

- strip() 152
- Traceback 248
- Übersicht 20
- upper() 144
- Veränderbare und unveränderbare Datentypen 106
- Verketten 20
- Wiederholung 20
- strip() 152
- strptime() 397, 398
- Style-Objekte 323
- sub() 184
- SUBJECT 427
- submit() 298
- submitButton 505
- submitButtonColor 505, 507
- subprocess 385, 405
- Subtraktion 17, 99
- Subtraktives Farbmodell 451
- Suchen
  - Befehlszeilenargumente 285
  - Browsertabs öffnen 287
  - E-Mails 425
  - Ergebnisse finden 286
  - Suchergebnisseite anfordern 285
  - Übersicht 285
  - Web 285
- sys.exit() 66
  
- T**
- tag\_name 296
- Tag-Objekte 283
- Tags 276
- Taskplaner 408
- Tastatur
  - Drücken und loslassen 500
  - String von der Tastatur senden 497
  - Tastenkombinationen 500
  - Tastennamen 498
- Tastenkombinationen 500
- Teillisten 92
- Telefonnummern
  - Reguläre Ausdrücke 187
  - Übereinstimmungen in der Zwischenablage finden 189
  - Übereinstimmungen zu einem String zusammensetzen 190
  - Übersicht 187
- Testfragengenerator 211
  - Anwortoptionen erstellen 214
  - Inhalt in Dateien schreiben 215
  - Reihenfolge der Fragen mischen 213
- Testdaten in Dictionary speichern 211
- Testfragen erstellen 213
- Übersicht 211
- text 351, 356
- text() 473
- TEXT 427
- Textausrichtung 150
- Textnachrichten
  - Automatische Benachrichtigungen 443
  - Nachrichten senden 440
  - Twilio 439
- textsize() 474
- Thread() 399, 404
- threading 385, 399
- Threads
  - Argumente übergeben 401
  - Downloadprogramm für Bilder 402
  - join() 405
  - Multithreading 399
  - Probleme der Nebenläufigkeit 402
  - Prozesse 405
  - Thread-Objekte 399
- Tic-Tac-Toe 128
- time
  - sleep() 387, 395
  - Stoppuhr 389
  - Übersicht 398
- time() 386
- timedelta 394, 398
- TLS-Verschlüsselung 421
- TO 427
- Top-Level-Domänen 189
- total\_seconds() 394, 398
- Traceback 248
- Transparenz 450, 460
- transpose() 463
- truetype() 474
- Truthy 60
- Tupel
  - tuple() 110
  - Übersicht 109
- Twilio
  - Automatische Textnachrichten 443
  - Modul 439
  - Textnachrichten senden 440
  - TwilioRestClient 441
  - Übersicht 440
- TypeError 91, 106
- typewrite() 497, 498, 502, 503, 509

**U**

Überschriften 360  
Ubuntu  
  cron 408  
  Dateien mit Standardanwendungen öffnen 409  
  Prozesse von Python aus starten 406  
  Unix-Philosophie 410  
Umbenennen 225  
  Dateien umbenennen 237  
  Datumsformate 233  
  Kalenderdaten in Dateinamen 233  
  Reguläre Ausdrücke für Kalenderdaten 234  
  Übersicht 233  
UNANSWERED 427  
UNDELETED 427  
underline 356  
UNDRAFT 427  
UNFLAGGED 428  
Ungleich 37  
Unicode 274  
Unix-Epoche 386, 392, 398  
Unix-Philosophie 410  
unlink() 227  
UNSEEN 427  
Unterstrich 23  
Unveränderbare Datentypen 106  
upper() 144  
UTC (Coordinated Universal Time) 386

**V**

ValueError 98, 397  
values() 122  
Variablen  
  Als Liste speichern 94  
  Benennung 23  
  Definition 21  
  Global 79  
  Initialisieren 22  
  Lokal 75  
  None 73  
  shelve 208  
  Überschreiben 22  
  Verweise 110  
  Zuweisungsanweisungen 21  
Veränderbare Datentypen 106  
Vergleichsoperatoren  
  Übersicht 37  
  Verwendung mit Binäroperatoren 41

Verizon 419, 424  
Verketten  
  Listen 93  
  Strings 19  
Verknüpfte Formate 354  
Verschachtelte Listen und Dictionarys 133  
Verschlüsselung 345  
Verweise  
  Übergeben 113  
  Übersicht 110  
Verzeichnisse  
  Absolute und relative Pfade 198  
  Aktuelles Arbeitsverzeichnis 197  
  Backslash und Schrägstrich 196  
  Definition 196  
  Durchlaufen 228  
  Endgültig löschen 227  
  Erstellen 199  
  Kopieren 224  
  os.path 200  
  send2trash 228  
  Trennzeichen 196  
  Umbenennen 225  
  Verschieben 225  
Volumes 196

**W**

Wahrheitswertetafeln 40  
walk() 229  
WARNING 255  
webbrowser (Modul)  
  Browser öffnen 268  
  open() 408  
WebDriver 295  
WebElement 295  
Web Scraping  
  Attribut abrufen 284  
  Auf Schaltflächen klicken 299  
  Bilder 288  
  bs4 281  
  Dateien speichern 274  
  Elemente finden 280, 295  
  Element mit select() finden 282  
  Entwickertools 278  
  Firefox 295  
  Formulare absenden 298  
  Google Maps 268  
  Google-Suche 285  
  HTML 275  
  Lernressourcen 276  
  Links folgen 297

Quelltext 277  
 requests 271  
 Selenium 294  
 Sondertasten 298  
 Übersicht 267  
 Webseiten 271  
 Websites vom Skript aus öffnen 408  
 Weißraum 152  
 Werte 16, 168  
 Wetterdaten 379  
     JSON-Daten herunterladen 380  
     JSON-Daten laden 381  
     Standort bestimmen 379  
     Übersicht 379  
 while-Schleifen  
     Endlosschleifen 488  
     Mauskoordinaten abrufen und ausgeben 488  
     Übersicht 51  
 Wiederholung  
     Listen 93  
     Strings 20  
 Windows  
     Dateien mit Standardanwendungen öffnen 409  
     Drittanbietermodule 513  
     Pfadtrennzeichen 196  
     pip 513  
     Prozesse von Python aus starten 406  
     Python-Programme ausführen 518  
     Taskplaner 408  
     Vom Automatisierungsprogramm abmelden 483  
 Word-Dokumente  
     Absatzformate 353  
     Bilder 361  
     Dokumente mit nicht standardmäßigen Formaten erstellen 355  
     Format 350  
     Lesen 351  
     python-docx 350  
     Run-Attribute 356  
     Schreiben 358  
     Text abrufen 352  
     Überschriften hinzufügen 360  
     Zeilen- und Seitenumbrüche 360  
 Workbook-Objekte 306  
 Worksheet-Objekte 306  
 write() 208  
 Writer-Objekte 371  
 writerow() 371

**X**  
 XKCD-Comics  
     Auf Threads warten 405  
     Bilder speichern 292  
     downloadXkcd() 403  
     Herunterladen 288  
     Multithread-Version 402  
     Programm 289  
     Threads erstellen und starten 404  
     Übersicht 288  
     Webseite herunterladen 291  
**Y**  
 Yahoo! Mail 419, 424  
**Z**  
 Zahlen raten 84  
 Zeichenformate 354  
 Zeichenklassen 178, 183  
 Zeichnen  
     Beispielprogramm 472  
     Ellipsen 472  
     ImageDraw 471  
     Linien 471  
     Polygone 472  
     Punkte 471  
     Rechtecke 472  
     Text 473  
 Zeilen (Excel)  
     Cell-Objekte abrufen 309  
     Höhe 327  
 Zeilenfortsetzungszeichen 105  
 Zeilenumbrüche 360  
 Zellen (Excel) 304  
     Cell-Objekte ansprechen 306  
     Verbinden und aufteilen 328  
     Werte schreiben 319  
 zipfile  
     Entpacken 232  
     Ordner 238  
     Übersicht 230  
     ZIP-Dateien erstellen 233  
     ZIP-Dateien lesen 231  
     ZipFile-Objekte 231  
 ZipInfo 231  
 Zirkumflex 183  
     Beginn eines Strings 180  
     Negative Zeichenklassen 180  
 Zusicherungen 249  
     Deaktivieren 519  
 Zuweisung 21, 39  
 Zwischenablage 270