2. Auflage

dpunkt.verlag

Aktualisiert auf Python 3

Al Sweigart

Routineaufgaben mit Python automatisieren

Praktische Programmierlösungen für Einsteiger

Al Sweigart

Routineaufgaben mit Python automatisieren

Praktische Programmierlösungen für Einsteiger

2., aktualisierte und erweiterte Auflage

dpunkt.verlag

Al Sweigart

Lektorat: Dr. Michael Barabas
Lektoratsassitenz: Anja Weimer
Fachgutachter/in: Ari Lacenski und Philip James
Übersetzung & Satz: G&U Language & Publishing Services GmbH, *www.gundu.com*
Copy-Editing: Ursula Zimpfer, Herrenberg
Herstellung: Stefanie Weidner
Umschlaggestaltung: Helmut Kraus, *www.exclam.de*
nach der Originalvorlage von No Starch Press
Druck und Bindung: mediaprint solutions GmbH, 33100 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 978-3-86490-753-1
PDF 978-3-96088-956-4
ePub 978-3-96088-957-1
mobi 978-3-96088-958-8

2., aktualisierte und erweiterte Auflage 2020

Wieblinger Weg 17
69123 Heidelberg

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: hallo@dpunkt.de.

5 4 3 2 1 0

Inhalt

Der Autor ..... **xxvi**

Der Fachgutachter ..... **xxvi**

Danksagung ..... **xxvii**

Einleitung **1**

Für wen ist dieses Buch gedacht? 2
Programmierstil 3
Was ist Programmierung? 3
- Was ist Python? 4
- Programmierer müssen nicht viel Mathe können 4
- Sie sind nie zu alt, um programmieren zu lernen 5
- Programmierung ist kreativ 6

Der Aufbau dieses Buchs 6
Python herunterladen und installieren 9
Mu herunterladen und installieren 10
Mu starten 10
IDLE starten 11
Die interaktive Shell 11
Drittanbietermodule installieren 12
Hilfe finden 13
Sinnvolle Fragen stellen 14
Zusammenfassung 16

Teil 1 Grundlagen der Python-Programmierung 17

1 Grundlagen von Python ... 19

Ausdrücke in die interaktive Shell eingeben ... 20
Die Datentypen für ganze Zahlen, Fließkommazahlen und Strings ... 23
Stringverkettung und -wiederholung ... 24
Werte in Variablen speichern ... 25
Zuweisungsanweisungen ... 25
Variablennamen ... 27
Ihr erstes Programm ... 28
Analyse des Programms ... 30
Kommentare ... 30
Die Funktion print() ... 31
Die Funktion input() ... 31
Den Benutzernamen ausgeben ... 32
Die Funktion len() ... 32
Die Funktionen str(), int() und float() ... 33
Zusammenfassung ... 36
Wiederholungsfragen ... 37

2 Flusssteuerung ... 39

Boolesche Werte ... 40
Vergleichsoperatoren ... 41
Boolesche Operatoren ... 43
Binäre boolesche Operatoren ... 43
Der Operator not ... 44
Kombinierte Verwendung von booleschen und Vergleichsoperatoren ... 45
Elemente zur Flusssteuerung ... 46
Bedingungen ... 46
Codeblöcke ... 46
Programmausführung ... 47
Flusssteuerungsanweisungen ... 47
If-Anweisungen ... 47
Else-Anweisungen ... 49
Elif-Anweisungen ... 50
While-Schleifen ... 57
Break-Anweisungen ... 61

Continue-Anweisungen 62
For-Schleifen und die Funktion range() 66
Module importieren 69
From-import-Anweisungen 71
Programme mit sys.exit() vorzeitig beenden 71
Ein kurzes Programm: Zahlen raten 72
Ein kurzes Programm: Stein, Schere, Papier 74
Zusammenfassung 78
Wiederholungsfragen 78

3 Funktionen 81

Def-Anweisungen mit Parametern 83
Terminologie 84
Rückgabewerte und die Anweisung return 84
Der Wert None 86
Schlüsselwortargumente und print() 87
Der Aufrufstack 88
Lokaler und globaler Gültigkeitsbereich 90
Lokale Variablen können im globalen Gültigkeitsbereich nicht verwendet werden 92
Lokale Gültigkeitsbereiche können keine Variablen aus anderen lokalen Gültigkeitsbereichen verwenden 92
Globale Variablen können von einem lokalen Gültigkeitsbereich aus gelesen werden 93
Lokale und globale Variablen mit demselben Namen 94
Die Anweisung global 95
Ausnahmebehandlung 97
Ein kurzes Programm: Zickzackmuster 99
Zusammenfassung 102
Wiederholungsfragen 102
Übungsprojekte 103
Die Collatz-Folge 103
Eingabevalidierung 104

4 Listen 105

Der Datentyp für Listen 106
Einzelne Elemente einer Liste mithilfe des Index abrufen 106

Negative Indizes 108
Teillisten mithilfe von Slices abrufen 108
Die Länge einer Liste mit len() abrufen 109
Werte in einer Liste mithilfe des Index ändern 109
Listenverkettung und -wiederholung 110
Elemente mit del aus einer Liste entfernen 110
Listen verwenden 110
Listen in for-Schleifen 112
Die Operatoren in und not in 113
Der Trick mit der Mehrfachzuweisung 114
Die Funktion enumerate() für Listen 115
Die Funktionen random.choice() und random.shuffle() für Listen 115
Erweiterte Zuweisungsoperatoren 116
Methoden 117
Elemente in einer Liste mit der Methode index() finden 117
Elemente mit den Methoden append() und insert() zu Listen hinzufügen 118
Elemente mit remove() aus Listen entfernen 119
Elemente in einer Liste mit sort() sortieren 119
Reihenfolge der Listeneinträge mit reverse() umkehren 121
Beispielprogramm: Magic 8 Ball mithilfe einer Liste 122
Sequenzielle Datentypen 122
Veränderbare und unveränderbare Datentypen 123
Der Datentyp für Tupel 125
Typen mit den Funktionen list() und tuple() umwandeln 126
Verweise 127
Identität und die Funktion id() 129
Verweise übergeben 130
Die Funktionen copy() und deepcopy() des Moduls copy 131
Ein kurzes Programm: Conways Spiel des Lebens 132
Zusammenfassung 137
Wiederholungsfragen 138
Übungsprojekte 139
Kommacode 139
Münzwurffolgen 139
Zeichenbildraster 140

5 Dictionaries und Datenstrukturen 143

Der Datentyp für Dictionaries 143
Dictionaries und Listen im Vergleich 144
Die Methoden keys(), values() und items() 146
Das Vorhandensein eines Schlüssels oder Wertes im Dictionary ermitteln 148
Die Methode get() 148
Die Methode setdefault() 149
Saubere Ausgabe 150
Datenstrukturen zur Modellierung realer Objekte 151
Ein Tic-Tac-Toe-Brett 153
Verschachtelte Dictionaries und Listen 158
Zusammenfassung 160
Wiederholungsfragen 160
Übungsprojekte 161
Validierer für Schach-Dictionary 161
Inventar für ein Fantasyspiel 161
Eine Funktion zum Hinzufügen von Listeninhalten zum Inventar-Dictionary 162

6 Stringbearbeitung 163

Umgang mit Strings 164
Stringliterale 164
Strings indizieren und Slices entnehmen 167
Die Operatoren in und not in für Strings 168
Strings in andere Strings einfügen 168
Nützliche Stringmethoden 169
Die Stringmethoden upper(), lower(), isupper() und islower() 169
Die isX-Stringmethoden 171
Die Stringmethoden startswith() und endswith() 173
Die Methoden join() und split() 173
Strings mit der Methode partition() aufteilen 175
Text mit rjust(), ljust() und center() ausrichten 176
Weißraum mit strip(), rstrip() und lstrip() entfernen 178
Die Funktionen ord() und chr() für numerische Zeichenwerte 178
Strings mit dem Modul pyperclip kopieren und einfügen 179

Projekt: Automatisierte Nachrichten mithilfe einer Mehrfach-Zwischenablage 180
Schritt 1: Programmdesign und Datenstrukturen 181
Schritt 2: Befehlszeilenargumente verarbeiten 181
Schritt 3: Die richtige Antwort kopieren 182
Projekt: Aufzählungspunkte zu einem Wiki-Markup hinzufügen 183
Schritt 1: Text von und zur Zwischenablage übertragen 184
Schritt 2: Textzeilen trennen und Sternchen hinzufügen 184
Schritt 3: Die veränderten Zeilen zusammenfügen 185
Ein kurzes Programm: Pig Latin 186
Zusammenfassung 190
Wiederholungsfragen 191
Übungsprojekt 192
Tabellenausgabe 192
Bots für Zombie Dice 192

Teil 2 Aufgaben automatisieren 197

7 Mustervergleich mit regulären Ausdrücken 199

Textmuster ohne reguläre Ausdrücke finden 200
Textmuster mithilfe regulärer Ausdrücke finden 203
Regex-Objekte erstellen 203
Vergleiche mit einem Regex-Objekt 204
Zusammenfassung: Mustervergleich mit regulären Ausdrücken 205
Weitere Möglichkeiten für den Mustervergleich mithilfe regulärer Ausdrücke 205
Gruppierung durch Klammern 205
Mithilfe der Pipe nach Übereinstimmungen mit mehreren Gruppen suchen 207
Optionale Übereinstimmung mit dem Fragezeichen 208
Mit dem Sternchen nach null oder mehr Übereinstimmungen suchen . 209
Mit dem Pluszeichen nach einer oder mehr Übereinstimmungen suchen 209
Mit geschweiften Klammern nach einer genauen Zahl von Wiederholungen suchen 210
Gieriger und nicht gieriger Mustervergleich 211
Die Methode findall() 212
Zeichenklassen 213

Eigene Zeichenklassen bilden 214
Zirkumflex und Dollarzeichen 214
Das Jokerzeichen 215
Beliebige Übereinstimmungen mit Punkt-Stern finden 216
Zeilenumbrüche mit dem Punktsymbol finden 217
Übersicht über Regex-Symbole 217
Übereinstimmungen ohne Berücksichtigung der Groß- und Kleinschreibung 218
Strings mit der Methode sub() ersetzen 218
Umgang mit komplizierten regulären Ausdrücken 219
Die Variablen re.IGNORECASE, re.DOTALL und re.VERBOSE kombinieren 220
Projekt: Extraktionsprogramm für Telefonnummern und E-Mail-Adressen 221
Schritt 1: Einen regulären Ausdruck für Telefonnummern erstellen . . . 222
Schritt 2: Einen regulären Ausdruck für E-Mail-Adressen erstellen . . . 223
Schritt 3: Alle Überstimmungen im Inhalt der Zwischenablage finden . 224
Schritt 4: Die gefundenen Übereinstimmungen zu einem String kombinieren 225
Das Programm ausführen 225
Ideen für ähnliche Programme 226
Zusammenfassung 226
Wiederholungsfragen 227
Übungsprojekte 229
Datumserkennung 229
Passwortstärke ermitteln 229
Regex-Version von strip() 229

8 Eingabevalidierung 231

Das Modul PyInputPlus 232
Die Schlüsselwortargumente min, max, greaterThan und lessThan . . . 234
Das Schlüsselwortargument blank 235
Die Schlüsselwortargumente limit, timeout und default 235
Die Schlüsselwortargumente allowRegexes und blockRegexes 236
Eine eigene Validierungsfunktion an inputCustom() übergeben 238
Projekt: Einen Trottel stundenlang beschäftigen 239
Projekt: Multiplikationstest 241
Zusammenfassung 243
Wiederholungsfragen 244

Übungsprojekte 244
Sandwichzubereiter 244
Ein eigenes Programm für den Multiplikationstest 245

9 Dateien lesen und schreiben 247

Dateien und Dateipfade 247
Backslash unter Windows und Schrägstrich unter macOS und Linux . 248
Pfade mit dem Operator / zusammenfügen 250
Das aktuelle Arbeitsverzeichnis 252
Das Benutzerverzeichnis 253
Absolute und relative Pfade 253
Neue Ordner mit os.makedirs() erstellen 254
Absolute und relative Pfade verwenden 255
Die Komponenten eines Dateipfads abrufen 257
Dateigrößen und Ordnerinhalte ermitteln 260
Eine Dateiliste mit Glob-Mustern bearbeiten 261
Die Gültigkeit von Pfaden prüfen 262
Dateien lesen und schreiben 263
Dateien mit der Funktion open() öffnen 265
Die Inhalte einer Datei lesen 266
Dateien schreiben 267
Variablen mit dem Modul shelve speichern 268
Variablen mit der Funktion pprint.pformat() speichern 270
Projekt: Zufallsgenerator für Tests 271
Schritt 1: Die Daten für den Test in einem Dictionary speichern 272
Schritt 2: Die Fragebogendatei erstellen und die Fragen mischen 273
Schritt 3: Die Auswahl der möglichen Antworten zusammenstellen . . 274
Schritt 4: Den Inhalt der Dateien für die Frage- und Lösungsbogen schreiben 275
Projekt: Aktualisierbare Mehrfach-Zwischenablage 276
Schritt 1: Kommentare und Vorbereitungen für die Shelf-Daten 277
Schritt 2: Den Inhalt der Zwischenablage unter einem Schlüsselwort speichern 278
Schritt 3: Schlüsselwörter auflisten und Inhalte laden 279
Zusammenfassung 280
Wiederholungsfragen 280

Übungsprojekte 281
Erweiterte Mehrfach-Zwischenablage 281
Lückentextspiel 281
Regex-Suche 282

10 Dateien verwalten 283

Das Modul shutil 284
Dateien und Ordner kopieren 284
Dateien und Ordner verschieben und umbenennen 285
Dateien und Ordner unwiederbringlich löschen 286
Sicheres Löschen mit dem Modul send2trash 287
Einen Verzeichnisbaum durchlaufen 288
Dateien mit dem Modul zipfile komprimieren 290
ZIP-Dateien lesen 291
ZIP-Dateien entpacken 292
ZIP-Dateien erstellen und Inhalte hinzufügen 293
Projekt: Amerikanische Datumsangaben in Dateinamen in europäische Datumsangaben ändern 293
Projekt: Amerikanische Datumsangaben in Dateinamen in 293
Schritt 1: Einen regulären Ausdruck für amerikanische Datumsangaben definieren 294
Schritt 2: Die einzelnen Teile der Datumsangabe in den Dateinamen ermitteln 296
Schritt 3: Die neuen Dateinamen zusammenstellen und die Dateien umbenennen 297
Vorschläge für ähnliche Programme 298
Projekt: Einen Ordner in einer ZIP-Datei sichern 298
Schritt 1: Den Namen der ZIP-Datei bestimmen 298
Schritt 2: Die neue ZIP-Datei erstellen 300
Schritt 3: Den Verzeichnisbaum durchlaufen und Inhalte zur ZIP-Datei hinzufügen 301
Vorschläge für ähnliche Programme 302
Zusammenfassung 302
Wiederholungsfragen 303
Übungsprojekte 303
Selektives Kopieren 303
Nicht mehr benötigte Dateien löschen 303
Lücken entfernen 304

11 Debugging 305

Ausnahmen auslösen 306
Traceback als String abrufen 308
Zusicherungen (Assertions) 309
Zusicherungen in einem Ampelsimulator 311
Protokollierung 312
Das Modul logging verwenden 313
Kein Debugging mit print() 315
Protokolliergrade 315
Die Protokollierung deaktivieren 317
Protokollierung in eine Datei 317
Der Debugger von Mu 318
Continue 318
Step In 319
Step Over 319
Step Out 319
Stop 319
Debugging eines Additionsprogramms 319
Haltepunkte 321
Zusammenfassung 323
Wiederholungsfragen 323
Übungsprojekt 324
Münzwurfprogramm 324

12 Web Scraping 325

Projekt: mapIt.py mit dem Modul webbrowser 326
Schritt 1: Die URL herausfinden 327
Schritt 2: Befehlszeilenargumente verarbeiten 327
Schritt 3: Den Inhalt der Zwischenablage verarbeiten und den Browser starten 328
Vorschläge für ähnliche Programme 329
Dateien mithilfe des Moduls requests aus dem Web herunterladen 329
Eine Webseite mit der Funktion requests.get() herunterladen 330
Auf Fehler prüfen 331
Heruntergeladene Dateien auf der Festplatte speichern 332
HTML 334
Quellen zu HTML 334
Ein kleiner Auffrischungskurs 334

Den HTML-Quellcode einer Webseite einsehen 335
Die Entwicklertools des Browsers öffnen 336
HTML-Elemente mithilfe der Entwicklertools finden 338
HTML mit dem Modul bs4 durchsuchen 340
Ein BeautifulSoup-Objekt aus dem HTML-Text erstellen 340
Elemente mit der Methode select() finden 341
Daten aus den Attributen eines Elements abrufen 344
Projekt: Alle Suchergebnisse öffnen 344
Schritt 1: Die Befehlszeilenargumente abrufen und die Suchergebnisseite anfordern 345
Schritt 2: Alle Ergebnisse finden 345
Schritt 3: Browsertabs für jedes Suchergebnis öffnen 346
Vorschläge für ähnliche Programme 347
Projekt: Alle XKCD-Comics herunterladen 348
Schritt 1: Den Aufbau des Programms festlegen 349
Schritt 2: Die Webseite herunterladen 350
Schritt 3: Das Bild des Comics finden und herunterladen 351
Schritt 4: Das Bild speichern und den vorherigen Comic suchen 352
Vorschläge für ähnliche Programme 353
Den Browser mit dem Modul selenium steuern 353
Einen seleniumgesteuerten Browser starten 354
Elemente auf der Seite finden 356
Auf Elemente klicken 358
Formulare ausfüllen und absenden 359
Die Betätigung von Sondertasten simulieren 359
Auf Browserschaltflächen klicken 360
Weitere Informationen über Selenium 361
Zusammenfassung 361
Wiederholungsfragen 361
Übungsprojekte 362
E-Mail-Programm für die Befehlszeile 362
Downloadprogramm für Fotowebsites 362
2048 363
Linküberprüfung 363

13 Excel-Arbeitsblätter 365

Excel-Dokumente 366
Das Modul openpyxl installieren 366
Excel-Dokumente lesen 367
Excel-Dokumente mit OpenPyXL öffnen 368
Arbeitsblätter aus der Arbeitsmappe abrufen 368
Zellen eines Arbeitsblatts abrufen 369
Umrechnen zwischen Kennbuchstaben und Nummern 370
Zeilen und Spalten eines Arbeitsblatts abrufen 371
Arbeitsmappen, Arbeitsblätter und Zellen 373
Projekt: Daten in einer Arbeitsmappe lesen 373
Schritt 1: Die Daten der Arbeitsmappe lesen 374
Schritt 2: Die Datenstruktur füllen 375
Schritt 3: Die Ergebnisse in eine Datei schreiben 377
Vorschläge für ähnliche Programme 378
Excel-Dokumente schreiben 379
Excel-Dokumente erstellen und speichern 379
Arbeitsblätter erstellen und entfernen 380
Werte in Zellen schreiben 381
Projekt: Ein Arbeitsblatt aktualisieren 381
Schritt 1: Eine Datenstruktur mit den neuen Informationen einrichten 382
Schritt 2: Alle Zeilen prüfen und die falschen Preise korrigieren 383
Vorschläge für ähnliche Programme 384
Die Schrift in den Zellen gestalten 384
Font-Objekte 385
Formeln 387
Das Erscheinungsbild von Zeilen und Spalten festlegen 388
Zeilenhöhe und Spaltenbreite festlegen 388
Zellen verbinden und aufteilen 389
Bereiche fixieren 390
Diagramme 391
Zusammenfassung 393
Wiederholungsfragen 394
Übungsprojekte 394
Multiplikationstabellen erstellen 395
Leere Zeilen einfügen 395
Zellen transponieren 396
Textdateien in Arbeitsblätter umwandeln 397
Arbeitsblätter in Textdateien umwandeln 397

14 Google Tabellen 399

EZSheets installieren und einrichten 399
Anmeldeinformationen und Tokendateien beziehen 400
Die Datei mit den Anmeldeinformationen widerrufen 402
Spreadsheet-Objekte 403
Tabellen erstellen, hochladen und auflisten 403
Tabellenattribute 405
Tabellen hoch- und herunterladen 406
Tabellen löschen 406
Sheet-Objekte 407
Daten lesen und schreiben 408
Tabellenblätter erstellen und löschen 412
Tabellenblätter kopieren 414
Grenzwerte für Google Tabellen 415
Zusammenfassung 415
Wiederholungsfragen 416
Übungsprojekte 416
Daten von Google Formulare herunterladen 416
Tabellen in andere Formate umwandeln 417
Fehler in einer Tabelle finden 417

15 PDF- und Word-Dokumente 419

PDF-Dokumente 419
Text aus PDFs entnehmen 420
PDFs entschlüsseln 422
PDFs erstellen 423
Projekt: Ausgewählte Seiten aus mehreren PDFs kombinieren 428
Schritt 1: Alle PDF-Dateien finden 429
Schritt 2: Die einzelnen PDFs öffnen 430
Schritt 3: Die einzelnen Seiten hinzufügen 430
Schritt 4: Die Ergebnisse speichern 431
Vorschläge für ähnliche Programme 432
Word-Dokumente 432
Word-Dokumente lesen 433
Den kompletten Text einer .docx-Datei abrufen 434
Absätze und Run-Objekte formatieren 435
Word-Dokumente mit anderen als den Standardformaten erstellen ... 437
Run-Attribute 438

Word-Dokumente schreiben 440
Überschriften hinzufügen 442
Zeilenwechsel und Seitenumbrüche hinzufügen 443
Bilder einfügen 443
PDF-Dateien aus Word-Dokumenten erstellen 444
Zusammenfassung 444
Wiederholungsfragen 445
Übungsprojekte 446
PDF-Paranoia 446
Personalisierte Einladungen als Word-Dokument 446
Brute-Force-Passwortknacker für PDFs 447

16 CSV-Dateien und JSON-Daten 449

Das Modul csv 450
Reader-Objekte 451
Daten in einer for-Schleife aus reader-Objekten lesen 452
Writer-Objekte 453
Die Schlüsselwortargumente delimiter und lineterminator 454
DictReader- und DictWriter-Objekte 455
Projekt: Kopfzeilen aus CSV-Dateien entfernen 457
Schritt 1: Alle CSV-Dateien durchlaufen 458
Schritt 2: Die CSV-Datei lesen 458
Schritt 3: Die CSV-Datei ohne die erste Zeile schreiben 459
Vorschläge für ähnliche Programme 460
JSON und APIs 461
Das Modul json 462
JSON-Daten mit der Funktion loads() laden 462
JSON-Daten mit der Funktion dumps() schreiben 463
Projekt: Die aktuellen Wetterdaten abrufen 463
Schritt 1: Den Standort aus dem Befehlszeilenargument entnehmen . . 464
Schritt 2: Die JSON-Daten herunterladen 465
Schritt 3: JSON-Daten laden und die Wettervorhersage ausgeben 466
Vorschläge für ähnliche Programme 467
Zusammenfassung 468
Wiederholungsfragen 468
Übungsprojekt 469
Excel-in-CSV-Konverter 469

17 Zeit und Aufgabenplanung 471

Das Modul time 472
Die Funktion time.time() 472
Die Funktion time.sleep() 473
Zahlen runden 474
Projekt: Superstoppuhr 475
Schritt 1: Das Programm auf die Zeitmessung vorbereiten 475
Schritt 2: Intervalldauern messen und anzeigen 476
Vorschläge für ähnliche Programme 477
Das Modul datetime 478
Der Datentyp timedelta 479
Anhalten bis zu einem bestimmten Zeitpunkt 481
datetime-Objekte in Strings umwandeln 481
Strings in datetime-Objekte umwandeln 483
Die Zeitfunktionen von Python im Überblick 483
Multithreading 484
Argumente an die Zielfunktion eines Threads übergeben 486
Probleme der Nebenläufigkeit 487
Projekt: Multithread-Version des XKCD-Downloadprogramms 488
Schritt 1: Eine Funktion für den Download verwenden 488
Schritt 2: Threads erstellen und starten 490
Schritt 3: Auf das Ende aller Threads warten 490
Andere Programme von Python aus starten 491
Befehlszeilenargumente an Popen() übergeben 494
Taskplaner, launchd und cron 494
Websites mit Python aufrufen 495
Andere Python-Skripte ausführen 495
Dateien in ihren Standardanwendungen öffnen 495
Projekt: Ein einfaches Countdown-Programm 496
Schritt 1: Der Countdown 497
Schritt 2: Die Klangdatei abspielen 497
Vorschläge für ähnliche Programme 498
Zusammenfassung 499
Wiederholungsfragen 500
Übungsprojekte 501
Elegantere Stoppuhr 501
Webcomic-Downloadprogramm mit Zeitplanung 501

18 E-Mails und Textnachrichten 503

E-Mails mit der Gmail-API senden und empfangen 504
Die Gmail-API aktivieren 505
E-Mails von einem Gmail-Konto senden 505
E-Mails in einem Gmail-Konto lesen 506
E-Mails in einem Gmail-Konto suchen 508
Anhänge von einem Gmail-Konto herunterladen 508
SMTP 509
E-Mails senden 510
Verbindung mit einem SMTP-Server aufnehmen 510
Die »Hallo«-Nachricht an den SMTP-Server senden 512
Die TLS-Verschlüsselung einleiten 512
Am SMTP-Server anmelden 513
Eine E-Mail senden 513
Die Verbindung zum SMTP-Server trennen 514
IMAP 514
E-Mails mit IMAP abrufen und löschen 515
Verbindung mit einem IMAP-Server aufnehmen 515
Am IMAP-Server anmelden 516
Nach E-Mails suchen 517
E-Mails abrufen und als gelesen markieren 521
E-Mail-Adressen aus einer Rohnachricht gewinnen 522
Den Rumpf aus einer Rohnachricht gewinnen 523
E-Mails löschen 524
Die Verbindung zum IMAP-Server trennen 524
Projekt: E-Mails über ausstehende Mitgliedsbeiträge senden 525
Schritt 1: Die Excel-Datei öffnen 526
Schritt 2: Alle säumigen Mitglieder finden 527
Schritt 3: Personalisierte E-Mail-Mahnungen senden 528
Textnachrichten über SMS-E-Mail-Gateways senden 529
Textnachrichten mit Twilio senden 531
Ein Twilio-Konto einrichten 532
Textnachrichten senden 532
Projekt: Das Modul »Just Text Me« 534
Zusammenfassung 535
Wiederholungsfragen 536

Übungsprojekte 537
Zufällige Zuweisung von Arbeiten 537
Regenschirmhinweis 537
Automatischer Entregistrierer 537
Den Computer per E-Mail steuern 538

19 Bildbearbeitung 541

Grundlagen zur Bilddarstellung auf Computern 542
Farben und RGBA-Werte 542
Koordinaten und Rechtecktupel 543
Bildbearbeitung mit Pillow 545
Mit dem Datentyp Image arbeiten 546
Bilder beschneiden 548
Bilder kopieren und in andere Bilder einfügen 549
Die Bildgröße ändern 552
Bilder drehen und spiegeln 553
Einzelne Pixel ändern 555
Projekt: Ein Logo hinzufügen 556
Schritt 1: Das Logobild öffnen 557
Schritt 2: Alle Dateien durchlaufen und die Bilder öffnen 558
Schritt 3: Die Bildgröße ändern 559
Schritt 4: Logo hinzufügen und Änderungen speichern 560
Vorschläge für ähnliche Programme 562
Bilder zeichnen 563
Formen zeichnen 563
Text zeichnen 565
Zusammenfassung 567
Wiederholungsfragen 568
Übungsprojekte 568
Das Logoprogramm erweitern und verbessern 569
Fotoordner auf der Festplatte finden 569
Personalisierte Tischkarten 570

20 GUI-Automatisierung 573

Das Modul PyAutoGUI installieren 574
Zugriff auf macOS erlauben 575

Kleine Probleme beheben 575
Die PyAutoGUI-Notfallsicherung 575
Beenden durch Abmelden 576
Den Mauszeiger steuern 576
Den Mauszeiger bewegen 577
Die Position des Mauszeigers abrufen 578
Mausinteraktionen 579
Klicken 579
Ziehen 579
Scrollen 582
Mausbewegungen planen 582
Auf dem Bildschirm arbeiten 584
Einen Screenshot aufnehmen 584
Einen Screenshot analysieren 584
Bilderkennung 586
Informationen über das Fenster abrufen 588
Das aktive Fenster abrufen 588
Andere Möglichkeiten zum Abrufen von Fenstern 589
Fenster bearbeiten 590
Die Tastatur steuern 592
Strings von der Tastatur senden 592
Tastennamen 594
Tasten drücken und loslassen 595
Tastenkombinationen 595
GUI-Automatisierungsskripte einrichten 596
Übersicht über die Funktionen von PyAutoGUI 597
Projekt: Formulare automatisch ausfüllen 599
Schritt 1: Den Ablauf herausfinden 601
Schritt 2: Die Koordinaten ermitteln 601
Schritt 3: Daten eingeben 603
Schritt 4: Auswahllisten und Optionsschalter 604
Schritt 5: Das Formular absenden und warten 605
Dialogfelder anzeigen 606
Zusammenfassung 608
Wiederholungsfragen 608

Übungsprojekte 609
Beschäftigung vortäuschen 609
Textfelder über Zwischenablage lesen 609
Instant-Messenger-Bot 610
Tutorial für einen Spiele-Bot 611

Anhang 613

A Drittanbietermodule installieren 613
Pip 613
Drittanbietermodule installieren 614
Module für Mu installieren 616

B Programme ausführen 619
Programme im Terminalfenster ausführen 619
Python-Programme unter Windows ausführen 621
Python-Programme unter macOS ausführen 622
Programme unter Ubuntu Linux ausführen 623
Python-Programme mit ausgeschalteten Zusicherungen ausführen 624

C Antworten auf die Wiederholungsfragen 625
Kapitel 1 626
Kapitel 2 626
Kapitel 3 628
Kapitel 4 629
Kapitel 5 630
Kapitel 6 630
Kapitel 7 631
Kapitel 8 632
Kapitel 9 632
Kapitel 10 633
Kapitel 11 633
Kapitel 12 634
Kapitel 13 635

Kapitel 14 636
Kapitel 15 636
Kapitel 16 637
Kapitel 17 637
Kapitel 18 638
Kapitel 19 638
Kapitel 20 639

Stichwortverzeichnis 641

Für meinen Neffen Jack

Der Autor

Al Sweigart ist Softwareentwickler und Fachbuchautor. Seine Lieblingsprogrammiersprache ist Python 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 11 Pfund.

Der Fachgutachter

Philip James arbeitet schon mehr als zehn Jahre mit Python und tritt in der Python-Community häufig als Redner zu unterschiedlichen Themen von Unix-Grundlagen bis zu Social Media auf Open-Source-Basis auf. Er ist einer der Hauptbeteiligten am BeeWare-Projekt und lebt mit seiner Partnerin Nic und deren Katze River in der Bay Area von San Francisco.

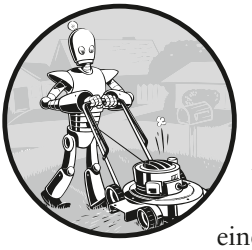
Danksagung

Es ist irreführend, dass nur mein Name auf dem Umschlag steht, denn um dieses Buch schreiben zu können, habe ich die Hilfe vieler Menschen benötigt. Ich möchte meinem Herausgeber Bill Pollock, meinen Lektoren Laurel Chun, Leslie Shen, Greg Poulos, Jennifer Griffith-Delgado und Frances Saux sowie allen anderen Mitarbeitern bei No Starch Press für ihre unschätzbare Hilfe danken. Ein Dank geht auch an meine Fachgutachter Ari Lacenski und Philip James für ihre hervorragenden Vorschläge und Verbesserungen sowie für ihre Unterstützung.

Vielen Dank auch allen Mitgliedern der Python Software Foundation für ihre großartige Arbeit. Die Python-Community ist die beste, die ich in der IT-Branche gefunden habe.

Schließlich möchte ich noch meiner Familie, meinen Freunden und der Clique bei Shotwell's danken, weil sie es mir nicht übel genommen haben, dass ich während des Schreibens an diesem Buch 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 eine Tabelle mit Tausenden von Produktpreisen anderer Läden auftauchte. Drei Mitarbeiter druckten diese Tabelle 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 der 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 hatte den Großteil dieser Stunden 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 auszuführen, ohne zu ahnen, dass der Computer diese Arbeit in wenigen Sekunden erledigen könnte, wenn er nur die richtigen Anweisungen dafür bekäme.

Für wen ist dieses Buch gedacht?

Software bildet die Grundlage vieler unserer Geräte, die wir für die Arbeit und in der Freizeit verwenden: Fast jeder nutzt soziale Netzwerke 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 Schulungen für Entwickler werden mit dem Versprechen beworben, ehrgeizige Anfänger zu Softwareingenieuren zu machen, die sechsstellige Gehälter verlangen können.

Dieses Buch ist jedoch nicht für diese Leute gedacht, 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 sind oder auch nur zur Arbeit oder zum Vergnügen einen Computer benutzen, 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 Sie 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 erleichtern. In diesem Buch lernen Sie, Wegwerfcode für einmalige 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 rasend schnell 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 genau 27 Mal.«

»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 schon 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* (»Zugriff 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 *https://python.org/* herunterladen, wobei es Versionen für Linux, macOS 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 die 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 9x9-Feldes die Zahlen von 1 bis 9 einfügen, wobei bereits einige Zahlen vorgegeben sind. Aus diesen Zahlen leiten Sie die Lösung durch Deduktion und Logik ab. In der Aufgabe in Abb. ***E–1*** kommt beispielsweise eine 5 sowohl in der ersten als auch in der zweiten Zeile vor. Daher muss die 5 im oberen rechten Quadrat in der dritten Zeile stehen. Da sich auch schon eine 5 in der letzten Spalte befindet, kann die 5 in der dritten Zeile nicht rechts neben der 6 stehen. Damit bleibt nur der Platz links von der 6 übrig. Jede Zeile, jede Spalte und jedes Quadrat, das Sie lösen, gibt Ihnen weitere Hinweise für den Rest des Rätsels. Mit jeder Gruppe der Zahlen von 1 bis 9, die Sie vervollständigen, nähern Sie sich der Lösung des gesamten Rätsels.

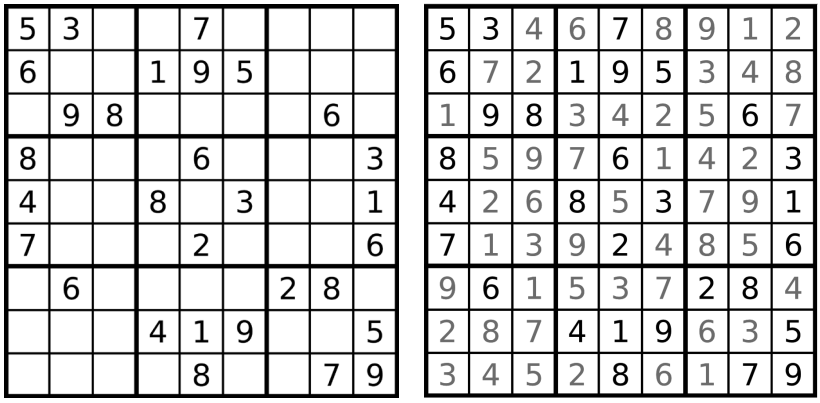


Abb. 2–1 *Ein Sudoku-Rätsel (links) und die Lösung (rechts). Beim Sudoku geht es zwar um Zahlen, doch ist dafür keine Mathematik erforderlich. (Bilder (c) Wikimedia Commons)*

Nur weil es bei Sudoku um Zahlen geht, heißt das noch lange nicht, dass man gut in Mathe sein muss, um die Lösung auszuknobeln. 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.

Sie sind nie zu alt, um programmieren zu lernen

Den zweithäufigsten Einwand, den ich im Zusammenhang mit Programmierung zu hören bekomme, ist, dass jemand glaubt, zu alt zu sein, um diese Tätigkeit noch zu erlernen. Im Internet musste ich schon viele Kommentare von Leuten lesen, die

meinten, sie seien mit sage und schreibe 23 Jahren (!) schon zu alt dafür. Das ist ganz sicher nicht »zu alt«, um programmieren zu lernen. Viele Menschen eignen sich in viel höherem Alter neue Fähigkeiten an.

Man muss nicht als Kind anfangen, um ein fähiger Programmierer zu werden. Die Vorstellung von Programmierern als Wunderkindern ist aber nicht totzukriegen. Leider habe ich selbst zu der Legende beigetragen, weil ich erzählt habe, dass ich bereits in der Grundschule mit dem Programmieren angefangen habe.

Programmieren lässt sich heutzutage leichter lernen als in den 90ern, denn heute gibt es viel mehr Bücher, bessere Suchmaschinen und viel mehr Websites, auf denen Sie Antworten auf Ihre Fragen bekommen. Vor allem aber sind die Programmiersprachen selbst viel benutzerfreundlicher geworden. Daher können Sie sich heute alles, was ich zwischen Grundschule und High-School-Abschluss über Programmierung gelernt habe, an ungefähr einem Dutzend Wochenenden erarbeiten. Mein Vorsprung war in Wirklichkeit gar kein so großer Vorsprung.

Programmieren lernt man nur durch Übung. Wir werden nicht als Programmierer geboren, und noch nicht programmieren zu können, heißt nicht, dass man niemals ein Experte auf diesem Gebiet werden könnte.

Programmierung ist kreativ

Programmieren ist eine ebenso kreative Tätigkeit wie Malen, Schreiben, Stricken und das Bauen mit Lego-Steinen. Genauso wie das Malen auf einer leeren Leinwand ist die Entwicklung von Software zwar gewissen Einschränkungen unterworfen, bietet aber auch unendlich viele Möglichkeiten.

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. Ein zehn Jahre alter Computer ist mehr als leistungsfähig genug, um damit Programme zu schreiben. Ein fertiges Programm können Sie beliebig oft kopieren. Während ein gestrickter Pullover immer nur von einer einzigen Person auf einmal getragen werden kann, lässt sich ein nützliches Programm auf einfache Weise der ganzen Welt online zur Verfügung stellen.

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 erwartet:

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: Eingabevalidierung Hier wird erläutert, wie Ihr Programm die von Benutzern eingegebenen Informationen überprüfen kann, damit die Daten nicht in einer Form vorliegen, die im weiteren Verlauf des Programms Fehler verursachen kann.

Kapitel 9: 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 10: 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 11: Debugging Hier werden die verschiedenen Instrumente vorgestellt, die in Python zur Verfügung stehen, um Fehler (Bugs) zu finden und zu beheben.

Kapitel 12: 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 13: Excel-Arbeitsblätter Hier geht es darum, wie Sie Excel-Arbeitsblätter programmgesteuert bearbeiten, sodass Sie das nicht manuell tun müssen. Das ist besonders praktisch, wenn die Anzahl der Dokumente, die Sie analysieren müssen, in die Hunderte oder gar in die Tausende geht.

Kapitel 14: Google Tabellen In diesem Kapitel erfahren Sie, wie Sie mit Python Daten in der häufig genutzten Online-Tabellenkalkulationsanwendung Google Tabellen lesen und bearbeiten.

Kapitel 15: PDF- und Word-Dokumente Dieses Kapitel behandelt das programmgesteuerte Lesen von Word- und PDF-Dokumenten.

Kapitel 16: CSV-Dateien und JSON-Daten Die Erklärung der programmgesteuerten Bearbeitung von Dokumenten wird hier für CSV- und JSON-Dateien fortgesetzt.

Kapitel 17: Zeit und Aufgabenplanung Hier lernen Sie, wie Python Uhrzeiten und Kalenderdaten handhabt und wie 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 18: E-Mails und Textnachrichten In diesem Kapitel geht es darum, Programme zu schreiben, die an Ihrer Stelle E-Mails und Textnachrichten senden.

Kapitel 19: Bildbearbeitung Dieses Kapitel erklärt, wie Sie Bilder, z. B. JPEG- oder PNG-Dateien, in Ihren Programmen bearbeiten können.

Kapitel 20: GUI-Automatisierung Hier lernen Sie, wie Sie mit einem Programm die Maus und die Tastatur steuern, um Mausklicks und Tastenbetätigungen zu simulieren.

Anhang A: Drittanbietermodule installieren Dieser Anhang zeigt, wie Sie Python mithilfe von nützlichen Modulen erweitern können.

Anhang B: Programme ausführen Hier erfahren Sie, wie Sie Python-Programme unter Windows, macOS und Linux außerhalb des Codeeditors ausführen.

Anhang C: Antworten auf die Wiederholungsfragen Hier finden Sie die Lösungen sowie einige zusätzliche Erklärungen zu den Wiederholungsfragen, die am Ende jedes Kapitels stehen.

Python herunterladen und installieren

Python können Sie kostenlos für Windows, macOS und Linux von *https://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. 3.8.0). 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 Downloadseite 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 macOS 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 in den nächsten Fenstern jeweils die Standardoptionen, indem Sie auf *Weiter* klicken.

Auf macOS laden Sie die passende *.dmg*-Datei für Ihre Version des Betriebssystems 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 Administratorpasswort eingeben.
2. Akzeptieren Sie in den nächsten Fenstern jeweils die Standardoptionen, indem Sie auf *Weiter* klicken, und klicken Sie auf *Agree*, um die Lizenzbedingungen zu akzeptieren.
3. Klicken Sie im letzten Fenster 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.

Mu herunterladen und installieren

Der *Python-Interpreter* ist die Software, die Ihre Python-Programme ausführt. Die Eingabe der Programme dagegen erfolgt im *Editor Mu* ähnlich wie in einer Textverarbeitung. Mu können Sie von *https://codewith.mu/* herunterladen.

Auf Windows und macOS laden Sie die Installerdatei für Ihr Betriebssystem herunter und führen sie aus, indem Sie darauf doppelklicken. Auf macOS wird dadurch ein Fenster geöffnet, in dem Sie das Mu-Symbol in den Ordner *Programm* ziehen müssen, um mit der Installation fortzufahren. Auf Ubuntu installieren Sie Mu als Python-Paket. Klicken Sie dazu auf der Downloadseite im Abschnitt *Python Package* auf die Schaltfläche *Instructions*.

Mu starten

Um Mu zu starten, gehen Sie wie folgt vor:

- Auf Windows 7 und höher klicken Sie auf das Startsymbol in der linken unteren Ecke, geben **Mu** in das Suchfeld ein und wählen die Anwendung aus.
- Auf macOS öffnen Sie ein Finder-Fenster, klicken auf *Programme* und dann auf *mu-editor*.
- Auf Ubuntu wählen Sie *Anwendungen* > *Zubehör* > *Terminal* und geben dann **python3 -m mu** ein.

Wenn Sie Mu zum ersten Mal ausführen, erscheint das Fenster *Select Mode* mit den Optionen *Adafruit CircuitPython*, *BBC micro:bit*, *Pygame Zero* und *Python 3*. Wählen Sie hier *Python 3*. Bei Bedarf können Sie den Modus später wieder ändern. Klicken Sie dazu auf die Schaltfläche *Mode* oben im Editorfenster.

Hinweis

Um die in diesem Buch verwendeten Drittanbietermodule installieren zu können, brauchen Sie Mu in der Version 1.1.0. Sie steht zurzeit als Alpha-Release unter einem eigenen Link getrennt von den anderen Downloadlinks zur Verfügung.

IDLE starten

In diesem Buch verwenden wir Mu als Editor und als interaktive Shell. Um Python-Code zu schreiben, können Sie jedoch auch beliebige andere Editoren verwenden. Zusammen mit Python wird die Software *IDLE* (*Integrated Development and Learning Environment*) installiert, die Sie ebenfalls als Editor verwenden können, wenn Sie Mu aus irgendeinem Grunde nicht installieren oder zum Laufen bekommen können. Um 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 macOS öffnen Sie ein Finder-Fenster, klicken auf *Programme*, dann auf *Python 3.8* und schließlich auf das IDLE-Symbol.
- Auf Ubuntu wählen Sie *Anwendungen* > *Zubehör* > *Terminal* und geben dann **idle3** ein. (Möglicherweise können Sie auch oben auf dem Bildschirm auf *Anwendungen* klicken, dann *Programming* auswählen und auf *IDLE3* klicken.)

Die interaktive Shell

Wenn Sie Mu ausführen, sehen Sie normalerweise das *Editorfenster*. Sie können in Mu allerdings auch eine *interaktive Shell* öffnen, indem Sie auf die Schaltfläche *REPL* klicken. Eine Shell ist ein Programm, in das Sie Anweisungen für den Computer eingeben können, ähnlich wie im Terminal von macOS oder an der Eingabeaufforderung von Windows. In die interaktive Python-Shell geben Sie die Anweisungen ein, die der Python-Interpreter ausführen soll. Der Computer liest diese Anweisungen und setzt sie sofort um.

In Mu ist die interaktive Shell ein Bereich in der unteren Hälfte des Fensters. Dort wird folgender Text angezeigt:

```
Jupyter QtConsole 4.3.1
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit
(AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Wenn Sie IDLE verwenden, erscheint als erstes Fenster die interaktive Shell. Sie ist bis auf den folgenden Text größtenteils leer:

```
Python 3.8.0b1 (tags/v3.8.0b1:3b5deb0116, Jun 4 2019, 19:52:55) [MSC v.1916
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Bei `[1]:` und `>>>` handelt es sich jeweils um die *Eingabeaufforderung*. In diesem Buch verwenden wir dafür das Zeichen `>>>`, da es gebräuchlicher ist. Auch im Terminal oder an der Windows-Eingabeaufforderung sehen Sie `>>>`. Die Eingabeaufforderung `[1]:` wurde für Jupyter Notebook erfunden, einen weiteren weitverbreiteten Python-Editor.

Zum Ausprobieren geben Sie an der Eingabeaufforderung der Shell Folgendes ein:

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

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

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

Damit haben Sie Ihrem Computer gerade eine Anweisung erteilt, und er hat genau das getan, was Sie von ihm verlangt haben!

Drittanbietermodule installieren

Für manche Zwecke muss ein Programm Module importieren. Einige davon sind im Lieferumfang von Python enthalten, aber andere – sogenannte Drittanbietermodule – wurden von Programmierern erstellt, die nicht zum Hauptentwicklerteam von Python gehören. In Anhang A finden Sie eine ausführliche Anleitung, um solche Module mit dem Programm `pip` (auf Windows) bzw. `pip3` (auf macOS und Linux) zu installieren. Wenn Sie in diesem Buch angewiesen werden, ein bestimmtes Drittanbietermodul zu installieren, schlagen Sie in Anhang A nach.

Hilfe finden

Programmierer neigen dazu, im Internet nach Antworten auf ihre Fragen zu suchen. Das ist eine ganz andere Art des Lernens, als viele es gewohnt sind. Es gibt hier keinen persönlich anwesenden Lehrer, der Ihnen etwas beibringt und Ihre Fragen beantwortet. Was das Internet als Klassenzimmer auszeichnet, ist die Tatsache, dass es dort sehr viele Menschen gibt, die Ihre Fragen beantworten können. Höchstwahrscheinlich ist Ihre Frage auch schon längst beantwortet worden, sodass die Lösung lediglich darauf wartet, dass Sie sie finden. Wenn Sie eine Fehlermeldung erhalten oder Schwierigkeiten haben, den Code das machen zu lassen, was er soll, sind Sie nicht die erste Person, die sich diesem Problem gegenübersieht. Daher ist es viel einfacher, als Sie glauben, eine Lösung zu finden.

Um Ihnen ein Beispiel zu geben, provozieren wir 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 danach. In diesem Fall also würden Sie **"TypeError: Can't convert 'int' object to str implicitly"** (in Anführungszeichen) in eine Suchmaschine eingeben. Daraufhin sehen Sie haufenweise Links, in denen erklärt wird, was diese Fehlermeldung bedeutet und was die Ursache ist (siehe ***E–2***).

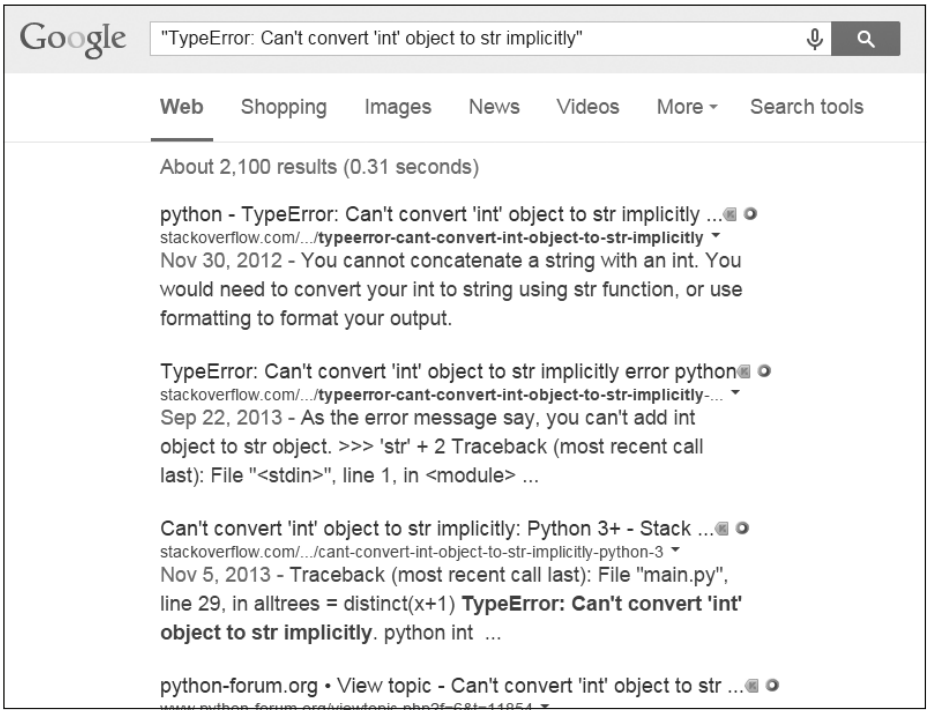


Abb. 2–2 *Google-Ergebnisse zu einer Fehlermeldung können sehr hilfreich sein.*

Sie werden dabei sehr oft bemerken, dass schon einmal jemand die gleiche Frage gestellt hat wie Sie und irgendeine hilfsbereite 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.

Sinnvolle Fragen stellen

Wenn Sie die Antworten auf Ihre Fragen nicht durch eine Onlinesuche finden können, versuchen Sie, Teilnehmer in Webforen wie Stack Overflow (*https://stackoverflow.com/*) oder dem Subreddit »Learn Programming« auf *https://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 *https://pastebin.com/* oder *https://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. Die 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 *https://pastebin.com/SzP2DbFx/* und *httpss://gist.github.com/asweigart/6912168/* ansehen.
- Erklären Sie, was Sie bereits versucht haben, um das 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.

Weitere Informationen darüber, wie Sie um Hilfe bei Programmierfragen bitten können, erhalten Sie in dem Blogpost auf *https://autbor.com/help/*. Eine Liste häufig gestellter Fragen über Programmierung finden Sie auf *https://www.reddit.com/r/learnprogramming/wiki/faq/*. Eine ähnliche Liste zu dem Thema, einen Job im Bereich der Softwareentwicklung zu bekommen, steht auf *https://www.reddit.com/r/cscareerquestions/wiki/index/*.

Ich helfe anderen gern dabei, Python kennenzulernen. So schreibe ich Programmiertutorials in meinem Blog auf *https://inventwithpython.com/blog/*. Sie können sich mit Ihren Fragen auch über *al@inventwithpython.com* an mich wenden (in Englisch). Eine schnellere Antwort dürften Sie allerdings erhalten, wenn Sie Ihre Fragen auf *https://reddit.com/r/inventwithpython/* stellen.

Zusammenfassung

Für die meisten Menschen ist ein Computer eher ein Gerät 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. Programmierung ist keine Gehirnchirurgie – Sie können sich auch als Anfänger daran versuchen und gefahrlos Fehler machen.

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, ist für Programmierer von unschätzbarem Wert.

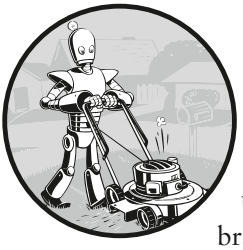
Fangen wir an!

Teil 1

Grundlagen der Python-Programmierung

1

Grundlagen von Python



Die Programmiersprache Python bietet 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 sich anzueignen, 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,* auch *REPL* (Read-Evaluate-Print Loop, also etwa »Lesen-Auswerten-Ausgeben-Schleife«) genannt, einzugeben. Damit können Sie eine Python-Anweisung nach der 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, können Sie den Editor Mu starten, den Sie beim Durcharbeiten der Installationsanleitungen im Vorwort heruntergeladen haben. Auf Windows öffnen Sie dazu das Startmenü, geben **Mu** ein und starten die gleichnamige Anwendung. Auf macOS öffnen Sie den Ordner *Programme* und doppelklicken darin auf *Mu*. Klicken Sie auf die Schaltfläche *New* und speichern Sie die leere Datei als *blank.py*. Wenn Sie diese leere Datei ausführen, indem Sie auf *Run* klicken oder F5 drücken, wird die interaktive Shell als neuer Bereich am unteren Rand des Mu-Fensters geöffnet. Dort sehen Sie die Eingabeaufforderung >>> der Shell.

Geben Sie dort **2 + 2** ein, um Python eine einfache Berechnung ausführen zu lassen. Das Mu-Fenster zeigt jetzt Folgendes an:

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

In Python wird etwas wie `2 + 2` als *Ausdruck* bezeichnet. Dies ist die einfachste Form von Programmieranweisungen 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. Daher können Sie an allen Stellen im Python-Code, an denen ein Wert stehen soll, 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/python_automatisieren_2/* 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
**	Exponent	`2 ** 3`	8
%	Modulo/Rest	`22 % 8`	6
//	Integerdivision/abgerundeter Quotient	`22 // 8`	2
/	Division	`22 / 8`	2.75
*	Multiplikation	`3 * 5`	15
-	Subtraktion	`5 - 2`	3
+	Addition	`2 + 2`	4

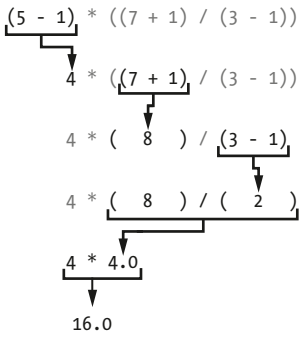
Tab. 1–1 *Arithmetische Operatoren, geordnet vom höchsten zum niedrigsten Rang*

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. Der Weißraum zwischen den Operatoren und Werten spielt in Python keine Rolle (außer bei den Einrückungen am Zeilenanfang). Ein Abstand von einem Leerzeichen ist jedoch üblich. 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 die folgende Grafik zeigt, wertet es dabei die einzelnen Teile eines Ausdrucks nacheinander aus, bis ein einziger Wert übrig ist:



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

Dies ist ein grammatikalisch korrekter deutscher Satz.

Dies grammatikalisch 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, wie die folgenden Beispiele zeigen:

```
>>> 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#0>", line 1, in <module>
    'Alice' + 42
TypeError: can only concatenate str (not "int") to str
```

Die Fehlermeldung `can only concatenate str (not "int") to str` 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
'AliceAliceAliceAliceAlice'
```

Der Ausdruck wird zu einem einzigen String 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>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

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. Wenn Sie das Ergebnis eines ausgewerteten Ausdrucks an einer späteren 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 Variablennamen, einem Gleichheitszeichen (das hier nicht als Gleichheitszeichen dient, sondern als *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. 1–1).

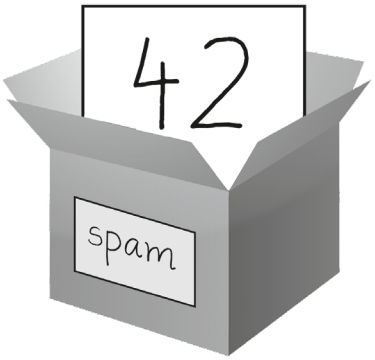


Abb. 1–1 *Die Anweisung* `spam = 42` *sagt dem Programm: »Die Variable* `spam` *enthält jetzt die Ganzzahl* 42.*«*

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), geht der alte Wert verloren. 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. 1–2 enthält die Variable `spam` in diesem Beispiel den Wert `Hello`, bis er durch `Goodbye` ersetzt wird.

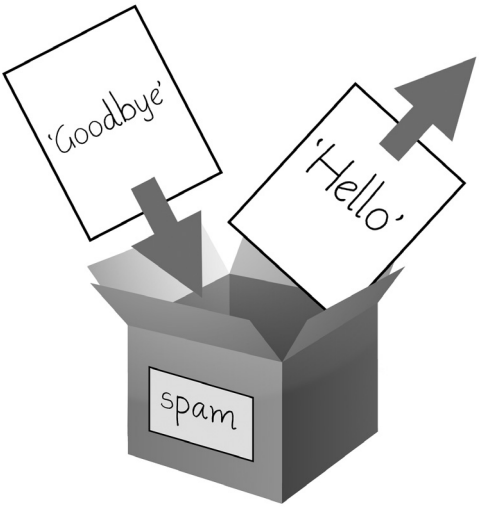


Abb. 1–2 *Wird einer Variablen ein neuer Wert zugewiesen, so wird der alte Wert vergessen.*

Variablennamen

Ein guter Variablenname beschreibt die enthaltenen Daten. 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.

Sie können Variablen in Python fast beliebig benennen, allerdings gibt es einige Einschränkungen. Tabelle 1–3 führt Beispiele für gültige Variablennamen auf. Sie müssen die folgenden drei Regeln erfüllen:

- Der Name muss ein einzelnes Wort sein, darf also keine Leerzeichen enthalten.
- Der Name darf nur aus Buchstaben, Ziffern und dem Unterstrich bestehen.
- Der Name darf nicht mit einer Zahl beginnen.

Gültige Variablennamen	Ungültige Variablennamen
`current_balance`	`current-balance` (Bindestriche sind nicht zulässig)
`currentBalance`	`current balance` (Leerzeichen sind nicht zulässig)
`account4`	`4account` (der Name darf nicht mit einer Zahl beginnen)
`_42`	`42` (der Name darf nicht mit einer Zahl beginnen)
`TOTAL_SUM`	`TOTAL_$UM` (Sonderzeichen wie `$` sind nicht zulässig)
`hello`	`'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. Zwar ist `Spam` ein zulässiger Variablenname, aber verabredungsgemäß sollten Variablennamen in Python mit einem Kleinbuchstaben beginnen.

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`. 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.«*

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 in Mu eine neue Datei anzulegen, klicken Sie in der obersten Zeile auf *New*.

In dem Fenster, das jetzt erscheint, sehen Sie einen Cursor, der auf Ihre Eingaben wartet. Dieses Fenster unterscheidet sich jedoch 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 Mu starten. Klicken Sie auf *Save*, geben Sie im Feld *File Name* den Namen **hello.py** 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 Mu beenden, verlieren Sie dann keinen Code. Als Tastaturkürzel zum Speichern einer Datei drücken Sie Strg + S auf Windows und Linux bzw. Cmd + S auf macOS.

Nachdem Sie das Programm gespeichert haben, führen Sie es aus. Drücken Sie dazu F5. Das Programm läuft jetzt im Fenster der interaktiven Shell. Beachten Sie aber, dass Sie F5 im Editorfenster drücken müssen, nicht im Shell-Fenster. Geben Sie Ihren Namen ein, wenn das Programm Sie danach fragt. Die Programmausgabe im Fenster der interaktiven Shell sieht wie folgt aus:

```
Python 3.7.0b4 (v3.7.0b4:eb96c37699, May 2 2018, 19:02:22) [MSC v.1913 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello world!
What is your name?
Al
It is good to meet you, Al
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.

Wie ein Programm ausgeführt wird, können Sie in dem Visualisierungsprogramm Python Tutor auf *http://pythontutor.com/* beobachten. Den Ablauf des vorhergehenden Programms finden Sie auf *https://autbor.com/hellopy/* zur Einsicht. Klicken Sie jeweils auf die Weiter-Schaltfläche und bewegen Sie sich durch die einzelnen Schritte der Programmausführung. Dabei können Sie beobachten, wie sich die Werte der Variablen und die Ausgabe ändern.

Analyse des Programms

Anhand Ihres neuen Programms im Dateieditor wollen wir uns genauer ansehen, was die einzelnen Programmzeilen bewirken und wozu die einzelnen Python-Anweisungen da sind.

Kommentare

Die erste 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 einer 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 die Codezeile wieder in das Programm aufnehmen möchten, können Sie das # einfach löschen.

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 in den Klammern angegebenen Stringwert 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 einen 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 Stringwertes.

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.

Das Klammernpaar hinter einem Namen zeigt 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 3 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
```

Sie können sich den Funktionsaufruf `input()` als einen Ausdruck vorstellen, der zu dem vom Benutzer eingegebenen String ausgewertet wird. Gibt der Benutzer beispielsweise **Al** ein, so wird der Ausdruck zu `myName = 'Al'` ausgewertet. In dem vorstehenden Code wird der Stringwert der Variablen `myName` zugewiesen.

Wenn Sie `input()` aufrufen und eine Fehlermeldung wie `NameError: name 'Al' is not defined` erhalten, liegt das Problem wahrscheinlich daran, dass Sie den Code mit Python 2 statt mit Python 3 ausführen.

Den Benutzernamen ausgeben

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

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

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

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:')    ❺
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 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 only concatenate str (not "int") to str
```

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 den 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 only concatenate str (not "int") to str
```

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 Wertes ausgewertet. Versuchen Sie in der interaktiven 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
```

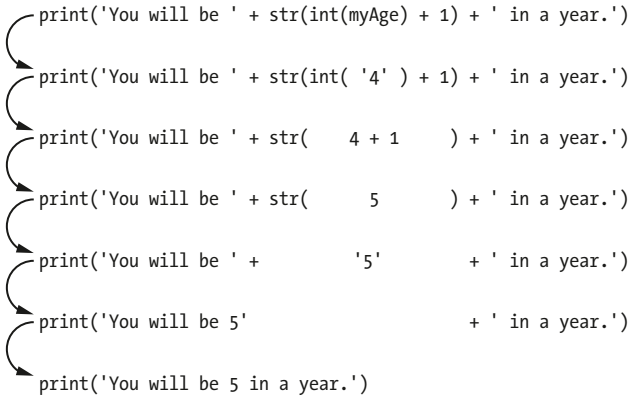
In Ihrem ersten Programm werden die Funktionen `int()` und `str()` in den letzten drei Zeilen verwendet, um im Code Werte des passenden Datentyps bereitzustellen.

```
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 wieder in einen String zurück, sodass er mit dem zweiten String `' in a year'` verkettet werden kann, um die endgültige Meldung zu bilden. Die Auswertung läuft wie folgt ab:



Das Verhältnis zwischen Text und Zahlen

Der Stringwert einer Zahl ist etwas völlig anderes als ihr Integer- oder Fließkommawert. Dagegen können Integer- und Fließkommawerte aber durchaus gleich sein.

```
>>> 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.

Zusammenfassung

Sie können arithmetische Ausdrücke mit einem Taschenrechner berechnen und Strings in einer Textverarbeitung verketten. Durch Kopieren und Einfügen können Sie sogar auf ganz einfache Weise eine Stringwiederholung erreichen. Ausdrücke, die aus Operatoren, Variablen und Funktionsaufrufen bestehen können, 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, Fließkommazahlen und Strings).

Sie haben auch schon einige Funktionen kennengelernt. `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 Wertes.

Im nächsten Kapitel lernen Sie, wie Sie in Python auf der Grundlage eines Wertes entscheiden, welcher Code ausgeführt, übersprungen oder wiederholt werden soll. Dies wird als *Flusssteuerung* bezeichnet. Damit können Sie Programme schreiben, die 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
```

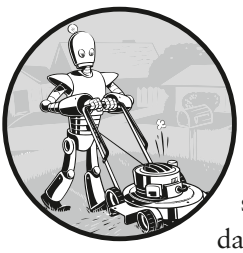
8. Warum ist `eggs` ein gültiger Variablenname, `100` dagegen nicht?
9. Mit welchen drei Funktionen können Sie die Integer-, Fließkomma- oder Stringversion eines Wertes ermitteln?
10. Warum ruft der folgende Ausdruck eine Fehlermeldung hervor? Wie können Sie ihn korrigieren?

```
'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 von Anweisungen und wissen, dass ein Programm nichts anderes als eine Abfolge solcher 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, welches Ergebnis die Auswertung eines Ausdrucks ergibt, kann das Programm entscheiden, Anweisungen zu überspringen oder zu wiederholen, oder unter mehreren möglichen Anweisungen auswählen. In der Praxis wird ein Programm so gut wie nie von der ersten bis zur letzten Anweisung einfach stur Zeile für Zeile ausgeführt. Stattdessen wird mithilfe von *Flusssteuerungsanweisungen* 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 Diagramm, um zu entscheiden, was zu tun ist, wenn es regnet. Folgen Sie dem Pfad entlang der Pfeile vom Anfang bis zum Ende.

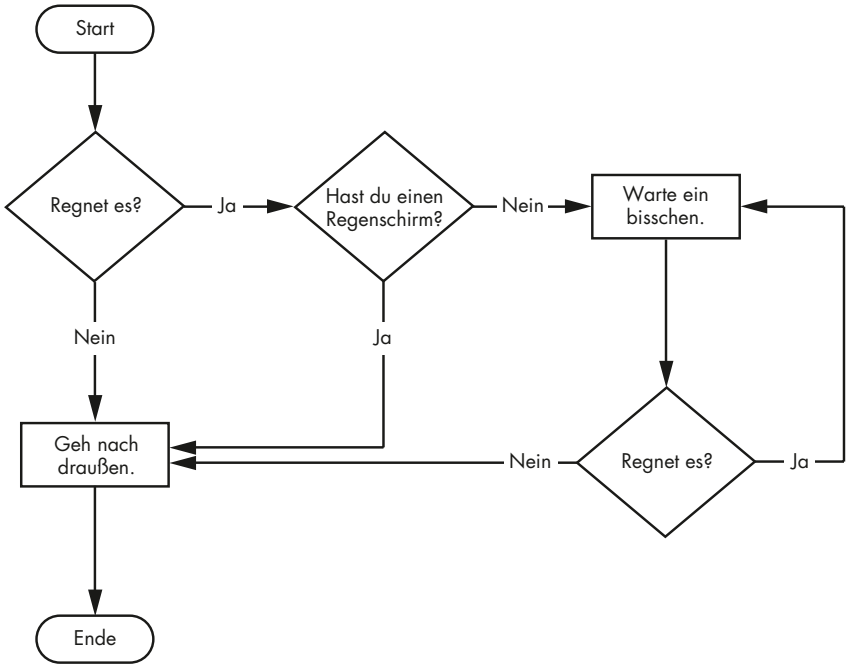


Abb. 2–1 *Ein Flussdiagramm, das Ihnen sagt, was Sie tun müssen, wenn es regnet.*

In einem Flussdiagramm gibt es gewöhnlich mehr als einen Weg, 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 verwendet, 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

Integer, Fließkommazahlen und Strings können unendlich viele mögliche Werte annehmen, doch für den *booleschen* Datentyp (benannt nach dem Mathematiker George Boole) gibt es 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 Wortes 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 den 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   ❶
True
>>> myAge = 29
>>> myAge >= 10   ❷
True
```