heiko KALISTA

PYTHON 3

Einsteigen und Durchstarten

Mit Kapitel zu MINECRAFT Pi

HANSER

Kalista

Python 3
Einsteigen und Durchstarten

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter

www.hanser-fachbuch.de/newsletter

Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter

www.hanser-fachbuch.de/update

Heiko Kalista

Python 3

Einsteigen und Durchstarten

HANSER

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

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.

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2018 Carl Hanser Verlag München, www.hanser-fachbuch.de
Lektorat: Sylvia Hasselbach
Copy editing: Walter Saumweber, Ratingen
Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-45469-9
E-Book-ISBN: 978-3-446-45689-1

Inhalt

Vorwort ..... **XV**

1 Grundlagen **1**

1.1 Hältst Du das richtige Buch in den Händen? 1
1.2 Dieses Buch bricht mit einigen Konventionen! 2
1.3 Die Arbeit mit diesem Buch 3
1.4 Das Kapitel zu Minecraft 4
1.5 Das Begleitmaterial zum Buch 4
1.6 Anregungen? Kritik? Forum? 5
1.7 Die Geschichte von Python in 120 Wörtern 5
1.8 Was kann man mit Python machen (und was nicht)? 6
1.9 Interpreter vs. Compiler 7
1.10 Python 2.7 oder 3.7? 8
1.11 Die Entwicklungsumgebung PyCharm 9
1.11.1 Alternativen zu PyCharm 10
1.12 Python-Interpreter installieren 10
1.12.1 Python-Interpreter unter Windows installieren 11
1.12.2 Python-Interpreter unter macOS installieren 12
1.12.3 Python-Interpreter unter Linux installieren 12
1.12.4 PyCharm unter Windows installieren 13
1.12.5 PyCharm unter macOS installieren 13
1.12.6 PyCharm unter Linux installieren 14
1.12.7 PyCharm einrichten 15
1.13 Genug geredet, los gehts! 17
1.13.1 Das erste Programm eingeben 18
1.13.2 Ausführen des ersten Beispiels 20
1.13.3 Laden der Beispiele 20
1.13.4 Der Quelltext im Detail 22
1.13.5 Kommentare im Detail 23
1.14 PEP8 – um sie ewig zu binden 25

1.15 Python-Programme ohne Entwicklungsumgebung starten ... 27
1.15.1 Python-Programme unter Windows starten ... 27
1.15.2 Python-Programme unter Linux oder macOS starten ... 28
1.16 Python interaktiv ... 29
1.17 Aufgabenstellung ... 31
1.18 Kurz & knapp ... 31

2 Variablen ... 33
2.1 Was sind Variablen? ... 33
2.2 Statisch typisiert vs. dynamisch typisiert ... 34
2.3 Einige Details vorab – Variablen sind auch nur Namen ... 36
2.4 Richtlinien für Variablennamen ... 38
2.4.1 Von Kamelen, Schlangen und Pascal ... 38
2.4.2 Sinnvolle Variablennamen ... 39
2.5 Rechnen mit Variablen ... 40
2.5.1 Verkürzte Schreibweise bei Rechenoperationen ... 41
2.5.2 Gibt es noch weitere Datentypen? ... 42
2.5.3 Konvertierung von Datentypen ... 43
2.5.4 Besonderheiten bei der Konvertierung von Datentypen ... 44
2.6 Formatierte Ausgabe mit print() ... 45
2.7 Genauigkeit von Fließkommazahlen ... 47
2.7.1 Formatierte Ausgabe von Fließkommazahlen ... 48
2.8 Den ganzzahligen Rest einer Division bestimmen ... 49
2.9 Konstanten, Konventionen und Restriktionen ... 50
2.10 Fehlerquelltext ... 51
2.10.1 Was ist die Aufgabe des Programms? ... 52
2.10.2 Lösungsvorschlag ... 52
2.10.3 Die gemeinen Kleinigkeiten ... 54
2.11 Aufgabenstellung ... 55
2.11.1 Einige Tipps ... 55
2.11.2 Lösung zur Aufgabenstellung ... 56
2.12 Kurz & knapp ... 57

3 Schleifen und Bedingungen ... 59
3.1 Was sind Schleifen und Bedingungen? ... 59
3.2 Die if-Bedingung ... 60
3.2.1 Blöcke und Einrückungen ... 61
3.2.2 Arbeit sparen mit else ... 62
3.2.3 elif ... 64
3.3 Der Platzhalter pass ... 66
3.4 Blöcke im interaktiven Modus ... 67
3.5 Logische Operatoren ... 67

3.6 PyCharm – Korrekturvorschläge übernehmen ... 70
3.7 while-Schleifen ... 71
3.7.1 Eine Schleife mit break vorzeitig beenden ... 72
3.7.2 continue in while-Schleifen ... 74
3.7.3 while-else ... 75
3.8 for-Schleifen ... 76
3.8.1 Eine einfache Zählschleife ... 77
3.8.2 Ein Wort zu Schleifenvariablen ... 78
3.8.3 Die Funktion range() im Detail ... 79
3.8.4 break und continue in for-Schleifen ... 80
3.8.5 Sonderfall Unterstrich: die Wegwerfvariable ... 81
3.9 Ein kurzer Abstecher: Exceptions ... 82
3.10 Fehlerquelltext ... 83
3.10.1 Was ist die Aufgabe des Programms? ... 84
3.10.2 Lösung zum Fehlerquelltext ... 85
3.11 Aufgabenstellung ... 86
3.11.1 Einige Tipps ... 87
3.11.2 Lösungsvorschlag ... 88
3.12 Ein umfangreicheres Beispiel: Zahlenraten ... 89
3.12.1 Die Hauptschleife ... 92
3.12.2 Ein neues Spiel starten ... 92
3.12.3 Abfrage des Schwierigkeitsgrades und Beenden des Spiels ... 93
3.13 Kurz & knapp ... 93

4 Funktionen ... 97
4.1 Was sind Funktionen? ... 97
4.2 Eine einfache Funktion definieren ... 97
4.3 Parameter, Argumente und Rückgabewert ... 99
4.4 Weitere Möglichkeiten, Funktionen aufzurufen ... 101
4.5 Globale und lokale Variablen ... 103
4.5.1 Ein bisschen mehr Verwirrung, bitte! ... 105
4.5.2 Das Schlüsselwort global ... 106
4.5.3 Auch Parameter sind lokal ... 108
4.5.4 Ein Wort zu globalen Variablen ... 109
4.5.5 Von Schatten und Hauptprogrammen ... 109
4.6 Standardwerte für Parameter ... 113
4.7 Schlüsselwortparameter ... 114
4.8 Wann sollten Funktionen zum Einsatz kommen? ... 116
4.9 Aufgabenstellung ... 117
4.9.1 Lösung zur Aufgabenstellung ... 118
4.10 Vorgehensweise bei größeren Projekten ... 121
4.11 Rekursion ... 122

4.12 Fehlerquelltext ... 123
4.12.1 Was ist die Aufgabe des Programms? ... 124
4.12.2 Lösung zum Fehlerquelltext ... 125
4.13 Kurz & knapp ... 127

5 Klassen ... 131
5.1 Was ist Objektorientierung? ... 131
5.2 Eine einfache Klasse definieren ... 132
5.2.1 Aufbau einer Klasse ... 133
5.2.2 Erzeugen von Objekten ... 134
5.2.3 Verwenden der Objekte ... 135
5.3 Kontrollierter Zugriff auf Attribute: Properties ... 136
5.3.1 Getter ... 136
5.3.2 Setter ... 139
5.3.3 Wann sollten Attribute und wann Properties verwendet werden? 140
5.3.4 Nachträgliches Umstellen auf Properties ... 141
5.4 Dynamische Attribute ... 142
5.5 Klassenattribute ... 145
5.5.1 Stolpersteine bei Klassenattributen ... 147
5.6 Statische Methoden ... 149
5.7 Zwischenstand: Braucht man das wirklich? ... 151
5.8 Aufgabenstellung ... 152
5.8.1 Einige Tipps ... 153
5.8.2 Lösungsvorschlag ... 154
5.9 Das Gleiche ist nicht dasselbe ... 156
5.9.1 Für Fortgeschrittene: Parameterübergabe im Detail ... 159
5.10 None ... 161
5.11 Vererbung ... 163
5.11.1 Ein einfaches Beispiel ... 164
5.11.2 Überschreiben von Methoden ... 166
5.12 Wie das Smartphone erfunden wurde – oder: Mehrfachvererbung ... 171
5.13 Für Fortgeschrittene: Binden von Methoden ... 174
5.13.1 Ein Blick hinter die Kulissen ... 174
5.13.2 Klassen um eigene Funktionen erweitern ... 176
5.13.3 Statische Methoden und instanzbezogene Bindung ... 178
5.14 Überschreiben der Methode __str__() ... 182
5.15 Und wo ist der Fehlerquelltext? ... 183
5.16 Kurz & knapp ... 183

6 Container ... 189
6.1 Was sind Container? ... 189
6.1.1 Eine Anmerkung, bevor es losgeht ... 189

6.2 Listen ... 190
6.2.1 Listen erzeugen und auf Elemente zugreifen ... 190
6.2.2 Listen dynamisch erzeugen ... 191
6.2.3 Elemente löschen oder ersetzen ... 193
6.2.4 Aufgabenstellung ... 196
6.2.4.1 Einige Tipps ... 197
6.2.4.2 Lösung zur Aufgabenstellung ... 197
6.2.5 Sortieren von Listen ... 198
6.2.5.1 Eigene Sortierfunktionen ... 200
6.2.5.2 Eine weitere Möglichkeit der Sortierung: sorted ... 202
6.2.5.3 Für Fortgeschrittene: lambda – anonyme Funktionen ... 203
6.2.6 Listen verknüpfen ... 206
6.2.7 Nicht nur für Fortgeschrittene: tiefes und flaches Kopieren ... 207
6.3 Tupel ... 209
6.3.1 Unterschiede zu Listen ... 209
6.3.2 Vorsicht: Instanzen in Tupeln können geändert werden ... 211
6.3.3 Mehrere Rückgabewerte mit Tupeln ... 212
6.3.4 Für Fortgeschrittene: namedtuple ... 214
6.4 Strings ... 216
6.4.1 Für Fortgeschrittene: Darum sind Strings unveränderlich ... 217
6.4.2 Slicing ... 219
6.4.3 Arbeiten mit Strings ... 221
6.4.4 Strings verbinden ... 221
6.4.5 Zerlegen und wieder zusammensetzen: split und join ... 223
6.4.6 Strings „aufbereiten“ ... 225
6.4.7 Ändern der Groß- und Kleinschreibung ... 227
6.4.8 Strings durchsuchen ... 228
6.4.9 Aufgabenstellung ... 230
6.4.9.1 Einige Tipps ... 231
6.4.9.2 Lösungsvorschlag ... 231
6.4.10 Ersetzungen durchführen ... 233
6.5 Dictionaries ... 235
6.5.1 Grundlagen von Dictionaries ... 235
6.5.2 Vorsicht beim Zugriff! ... 236
6.5.3 Dictionaries durchlaufen und verändern ... 238
6.5.4 Elemente aus einem Dictionary entfernen ... 240
6.6 Fehlerquelltext ... 242
6.6.1 Was ist die Aufgabe des Programms? ... 242
6.6.2 Lösung zum Fehlerquelltext ... 243
6.7 Sets und Frozensets ... 245
6.7.1 Einfache Sets/Frozensets erzeugen ... 245
6.7.2 Sets: Elemente hinzufügen und entfernen ... 247
6.7.3 Mengenoperationen ... 248
6.8 Kurz & knapp ... 252

7 Exceptions ... 261
7.1 Was sind Exceptions? ... 261
7.2 Fehler? Die passieren mir doch nicht! ... 261
7.3 Die Mechanik von Exceptions ... 263
7.4 Abfangen unterschiedlicher Exceptions ... 265
7.5 Alle Exceptions fangen ... 267
7.6 Eigene Exceptions ... 269
7.7 else und finally ... 272
7.8 Einige allgemeine Tipps ... 275
7.9 Kurz & knapp ... 279

8 Module und Pakete ... 283
8.1 Module ... 283
8.1.1 Grundlagen ... 284
8.1.2 Dokumentation von Modulen ... 286
8.1.3 Umbenennen eines Namensraums ... 288
8.1.4 Selektives Importieren ... 289
8.1.5 Namensräume haben ihren Sinn ... 290
8.1.6 Batteries included! ... 290
8.1.7 Reihenfolge beim Importieren ... 291
8.1.8 Eigene Module in mehreren Projekten nutzen ... 292
8.1.9 Tipps für das Schreiben von Modulen ... 292
8.1.10 Automatische Ausführung beim Import verhindern ... 294
8.2 Pakete ... 295
8.2.1 Importieren von Modulen aus Paketen ... 296
8.2.2 Reguläre Pakete ... 297
8.2.3 Namespace Packages ... 298
8.2.4 Unterscheidung der Paketarten ... 299
8.3 Kurz & knapp ... 300

9 Dateien und Dateisystem ... 303
9.1 Lesen und Schreiben von Dateien ... 303
9.1.1 Eine einfache Textdatei auslesen ... 304
9.1.2 Fehler beim Öffnen von Dateien ... 305
9.1.3 Eine Datei schrittweise auslesen ... 306
9.1.4 Zeilenweises Auslesen von Textdateien ... 308
9.1.5 Textdateien schreiben ... 309
9.1.6 Übersicht möglicher Modi der Funktion open() ... 311
9.1.7 Aufgabenstellung ... 312
9.2 Dateien und Dateisystem ... 312
9.2.1 Verzeichnisse ... 313
9.2.2 Pfade und Prüfung auf deren Existenz ... 314
9.2.3 Pfade und Plattformunabhängigkeit ... 316

9.2.4 Verzeichnisse und Dateien erzeugen und löschen ... 317
9.2.5 Verzeichnisstrukturen erzeugen und löschen ... 320
9.2.6 Umbenennen von Verzeichnissen und Dateien ... 322
9.2.7 Kopieren und Verschieben ... 324
9.3 Kurz & knapp ... 326

10 GUI-Programmierung mit tkinter ... 331
10.1 Warum gerade tkinter? ... 331
10.2 Die Layout-Manager ... 332
10.2.1 Absolute Positionierung ... 332
10.2.2 Der pack-Manager ... 333
10.2.3 Der grid-Manager ... 338
10.2.4 Welcher Manager soll es sein? ... 340
10.3 Flexible Fenstergröße ... 341
10.4 Konfiguration von Widgets ... 344
10.5 Steuerelemente im Detail ... 346
10.5.1 Buttons ... 346
10.5.2 Kontrollvariablen ... 348
10.5.3 Eingabefelder ... 349
10.5.4 Textboxen ... 350
10.5.5 Checkboxen ... 354
10.5.6 Radiobuttons ... 356
10.5.7 Aufgabenstellung ... 357
10.5.8 Lösungsvorschlag ... 358
10.5.9 Listboxen ... 359
10.5.10 Listbox mit Scrollbalken ... 362
10.6 Fehlerquelltext ... 363
10.6.1 Was ist die Aufgabe des Programms? ... 363
10.6.2 Lösung zum Fehlerquelltext ... 364
10.7 Dialogfenster ... 366
10.8 Menüs ... 369
10.8.1 Einfache Menüs ... 369
10.8.2 Untermenüs ... 370
10.9 Kurz & knapp ... 372

11 Debugging ... 377
11.1 Was ist ein Debugger? ... 378
11.2 Eine einfache Fehlersuche ... 378
11.2.1 Haltepunkte setzen und entfernen ... 379
11.2.2 Das Programm durchlaufen und Werte betrachten ... 380
11.2.3 Geht es auch ohne Haltepunkte? ... 383
11.2.4 Interaktive Fehlersuche mit der Python-Konsole ... 384
11.3 Debuggen von Funktionen und Methoden ... 386

11.3.1 In Funktionen springen . . . 387
11.3.2 Abschnitte überspringen und Funktionen verlassen . . . 388
11.3.3 Clever springen . . . 389
11.4 Watches . . . 390
11.5 Haltepunkte im Detail . . . 392
11.5.1 Ein Beispiel zum Experimentieren . . . 392
11.5.2 Verwalten von Haltepunkten . . . 393
11.5.3 Unterbrechen oder nicht? . . . 397
11.5.4 Bedingte Haltepunkte . . . 397
11.5.5 Protokollierung . . . 398
11.5.6 Temporäre Haltepunkte . . . 399
11.5.7 Verkettete Haltepunkte . . . 400
11.5.8 Haltepunkte für Exceptions . . . 400
11.5.9 Kombination der Optionen . . . 401
11.6 Einsatz in der Praxis . . . 402

12 Versionsverwaltung mit Git . . . 403
12.1 Der vermeintlich leichte Weg . . . 403
12.2 Wie funktionieren Versionskontrollsysteme? . . . 404
12.2.1 Verallgemeinerte Arbeitsweise . . . 404
12.2.2 Zentrale und verteilte Versionskontrolle . . . 405
12.3 Was ist Git? . . . 405
12.4 Interessiert mich nicht, ich arbeite alleine! . . . 407
12.5 Vorbereitungen . . . 407
12.5.1 Git unter Windows installieren . . . 407
12.5.2 Git unter macOS installieren . . . 408
12.5.3 Git unter Linux installieren . . . 409
12.5.4 GitHub . . . 409
12.6 Los geht's – Git lokal verwenden . . . 410
12.6.1 Ein bestehendes Projekt unter Versionskontrolle stellen . . . 410
12.6.2 Dateien hinzufügen . . . 412
12.6.3 Commit/Einchecken . . . 413
12.6.4 Änderungen vornehmen und überprüfen . . . 414
12.6.5 Änderungen rückgängig machen . . . 416
12.6.6 Betrachten der Historie . . . 417
12.6.7 Zu älteren Versionen zurückkehren – Möglichkeit 1 . . . 419
12.6.8 Dateien ignorieren . . . 420
12.7 Zusammenarbeit über ein Remote Repository . . . 421
12.7.1 Projekt auf GitHub veröffentlichen . . . 422
12.7.2 Ein Repository klonen . . . 422
12.7.3 Änderungen pushen . . . 424
12.7.4 Pull . . . 424
12.7.5 Wer hat's erfunden? . . . 425

12.7.6 Automatisches Merging . . . 425
12.7.7 Mergen und Konflikte beheben . . . 426
12.8 Branching . . . 430
12.8.1 Um was geht es? . . . 430
12.8.2 Einen Branch erzeugen und damit arbeiten . . . 431
12.8.3 Zwischen Branches wechseln . . . 433
12.8.4 Branches mergen . . . 433
12.8.5 Branches pushen . . . 435
12.8.6 Fetch . . . 435
12.8.7 Zu älteren Versionen zurückkehren – Möglichkeit 2 . . . 436
12.9 Weitere nützliche Features . . . 436
12.9.1 Stashing – Änderungen temporär speichern . . . 437
12.9.2 Commits korrigieren . . . 439
12.9.3 Tagging . . . 439
12.10 Noch ein paar Tipps . . . 440
12.10.1 Zwei einfache Branching-Modelle . . . 440
12.10.2 Atomare Commits . . . 441
12.10.3 Test-Repository griffbereit halten . . . 442
12.10.4 Andere Git-Clients . . . 442
12.11 Kurz & knapp . . . 443

13 Minecraft auf dem Raspberry Pi . . . 447
13.1 Um was geht es in diesem Kapitel? . . . 447
13.1.1 Der Raspberry Pi – ein kleines Kraftpaket . . . 447
13.1.2 Minecraft Pi . . . 448
13.1.3 Was wird benötigt? . . . 449
13.2 Der Sprung ins kalte Wasser . . . 450
13.3 Die Entwicklungsumgebungen . . . 450
13.4 Den Raspberry Pi einrichten . . . 451
13.4.1 Erstellen der SD-Karte . . . 451
13.4.2 Einstellen des Tastaturlayouts . . . 452
13.4.3 Minecraft starten . . . 452
13.4.4 Die Steuerung . . . 452
13.5 Der erste Testlauf . . . 454
13.5.1 Arbeiten mit IDLE . . . 454
13.5.2 Arbeiten mit Thonny . . . 456
13.6 Das Begleitmaterial . . . 457
13.7 Die Minecraft Python API . . . 457
13.7.1 Quellen und weitere Informationen: . . . 458
13.7.2 Eine Übersicht der Minecraft API . . . 458
13.7.3 Die Klasse Minecraft . . . 458
13.7.4 Die Klasse CmdCamera . . . 461
13.7.5 Die Klasse CmdPlayer . . . 462

13.7.6 Die Klasse CmdEntity 462
13.7.7 Die Klasse Block 463
13.7.7.1 Eine Übersicht der wichtigsten Blöcke 463
13.7.8 Noch einmal alles zusammen 464
13.8 Beispiele zu Schleifen und Bedingungen 466
13.8.1 Blocktypen ausprobieren 466
13.8.2 Spielfigur automatisch durch die Welt bewegen 468
13.8.3 Eine Treppe bauen 469
13.8.4 Eine Pyramide bauen 470
13.9 Beispiele zu Funktionen 472
13.9.1 Swimmingpools bauen 472
13.9.2 Moderne Kunst? 474
13.10 Beispiele zu Klassen 477
13.10.1 Blöcke regnen lassen 477
13.10.2 Blinklichter 480
13.11 Beispiele zu Containern 483
13.11.1 Lichterkette 483
13.11.2 Mengenoperationen 485
13.12 Ein Beispiel zu Modulen und Paketen 487
13.13 exit() – wie geht es weiter? 490

Index 491

Vorwort

Wer sich in die Welt der Programmierung wagt, steht meist vor der Frage, welche Programmiersprache am besten für den Einstieg geeignet ist. Die Antwort darauf hängt naturgemäß von verschiedenen Aspekten ab: Was möchte man entwickeln, welche Hardware und welche Betriebssysteme spielen eine Rolle und wie leicht ist die gewählte Sprache zu lernen?

Diese Fragen stellen sich jedoch nicht nur dann, wenn man sich zum ersten Mal mit der Programmierung beschäftigt und einen Einstieg sucht. Als Entwickler sieht man sich häufig mit neuen Situationen konfrontiert und muss abwägen, ob nicht eine andere Programmiersprache als die bisher verwendete in der aktuellen Situation besser geeignet ist. Eine Programmiersprache ist wie ein Werkzeug: Man sucht sich für jede Aufgabe das passende heraus. Doch es gibt auch einen weiteren, viel einfacheren Grund, eine neue Programmiersprache zu lernen: die reine Neugierde, die den meisten Entwicklern eigen ist.

Python erfreut sich nicht ohne Grund großer und stets steigender Beliebtheit. Schließlich handelt es sich um eine Sprache, die sich leicht erlernen lässt und mit der gleichzeitig enorm viel möglich ist. Ob Du nun bereits eine Programmiersprache beherrschst, oder ob Du Dich zum Einstieg für Python entschieden hast: Dieses Buch ist so konzipiert, dass Du es in beiden Fällen nutzen kannst.

Beim Schreiben dieses Buches war es mir besonders wichtig, eine lockere und gemütliche Atmosphäre zu erzeugen. Wenn Du beim Lesen den Eindruck hast, dass man bei einer Tasse Kaffee zusammensitzt und gemeinsam neue Themen entdeckt, dann ist mir das hoffentlich auch gelungen.

Danksagung

Der erste und größte Dank gilt an dieser Stelle Naomi, die immer an meiner Seite ist und jederzeit Verständnis hatte, wenn ich mich zum Schreiben ins stille Kämmerlein zurückzog. Ohne Deine Unterstützung wäre dieses Buch niemals zustande gekommen!

Besonderer Dank gilt allen meinen Freunden und meiner Familie für das nicht selbstverständliche Verständnis in hektischen Zeiten.

Bettina Zankl möchte ich für das Durcharbeiten der einzelnen Kapitel und für die vielen konstruktiven und wertvollen Vorschläge danken. Der nächste Behelfsdöner kann kommen!

Sylvia Hasselbach, Irene Weilhart, und Kristin Rothe vom Carl Hanser Verlag möchte ich für die tolle Zusammenarbeit danken. Ich wette, irgendwo im Verlagsgebäude hängt nun ein Schild mit dem Aufdruck „Herr Kalista hat mal wieder eine Frage …“.

Vielen Dank an Walter Saumweber für das gewissenhafte Korrektorat und das tolle Feedback!

Am Rande Hessens, im Juli 2018

Heiko Kalista

```
SEAABL AANISE LZTTEE ELIIEI APRNEN DRAAMT
ETSRVU STTIRC NSUESN BHHLZG NHAEUG ENOHOE
INMODE NSEPKG
----------------------------------------
?!#:!! !-?#?: !?!:!- !?!::? -!-?#? #?!:?:
!!!::: :?!!:# !:!!!? !#!#?: !#-!:- ?:?!-:
:--!?! ?!!!?: :?:!?! ?#:!-# :-!?:- --???!
!:-?-? !!#:?- :?#::# #?!:#: ?-##!: ?!?!?!
:?!-:? !-::::? !?!?!# #?!:?? ::-:-- ?-#!!?
#:#--# #?:-#- ???!-- ?!::#! :!:#:: #-?!:#
!-##:! !!:-:: -!?!?! -!:!:! :?::?! ?::?!:|
--!?!: !-#?!: !##:-: :???:: ?:!#!: !!:?#:
:!-#?? ?:-?!- ?-#::? !-:-:- !!-??- --:-#?
?!?:#- ::!!?! !#::?: ?-!?-# ?!!#?# :-:#-!
:!#::- ##?#?! -:???? -?#-!! :?-!?- ?:?#?:
???!!: -??!-: #!!??! !?#:?! ##::?? ???!!-
#??!?!

!Apfeltäubling#DornigerStachelbart?Flocke
nstieligerHexenröhrling:GelbeLohblüte-Xer
ocomellusChrysenteron
----------------------------------------
RDEKYY IXEYGD ICYHHZ CVUOPI NYRQAR IRRQYU
EJHVGM GXMISF VBLEIR PHRHFQ UJMUFU MBIAVV
URKHAL YDXLIF HIXREJ DSALUC GEHGUM JVEUHP
LMNQVF PCJEWX FBLXZQ YMWOQI QIDVBV CRXYYH
GXEULM ALLIQP REXYYY VGMVRD IWIIYH YHZOQV
LGBVLL GQOHVB QYJNCY XJQDYU WHDSAH BMOUIS
YNOLTR RMLYAJ INISKY JECJNM AZWPYP HRTIHF
XIDLHS GIAPMM HVNGDE BVYVXM UVSQOS EXQFEG
VQURQD CCWEIH LWIAGE AUIUWG DTCZMI ZIVYHY
EUHSRP MGUCCV AFRIEZ KUDQDM FATFBZ CLWXAW
MEIGTE CEXVXT CCM
----------------------------------------
                        geloest@icloud.com
```

1 Grundlagen

1.1 Hältst Du das richtige Buch in den Händen?

Du möchtest die Programmiersprache Python erlernen oder Dein bereits vorhandenes Wissen darin erweitern? Dann bist Du hier genau richtig! Egal ob Du Dich zum ersten Mal mit dem Thema Programmierung auseinandersetzt oder schon Erfahrungen mit anderen Sprachen hast: Dieses Buch wird Dir dabei helfen, Python effektiv zu nutzen. Bereits im ersten Kapitel wirst Du erste Erfolge erzielen und Ergebnisse sehen, ohne Dich zuvor durch einen Theorie-Dschungel kämpfen zu müssen. Die einzigen Voraussetzungen, die Du mitbringen musst, sind Grundkenntnisse im Umgang mit Computern, Experimentierfreudigkeit, Neugier und natürlich auch Geduld. In diesem Buch wird zwar Wert auf einen praxisnahen Einstieg gelegt, aber das bedeutet nicht, dass Du innerhalb weniger Tage besonders ausgeklügelte Anwendungen entwickeln wirst. Das Lernen einer Programmiersprache ist ein stetiger Prozess, der, wie so vieles andere auch, aus Hochs und Tiefs besteht. Ich möchte Dir daher nicht das Versprechen geben, dass Du in x Tagen das Programmieren erlernst. Dafür versichere ich Dir, dass sämtliche Themen mit der nötigen Ausführlichkeit besprochen werden und es immer Tipps und Hilfestellungen geben wird.

Es spielt keine Rolle, welches Betriebssystem Du bevorzugst. Du kannst mit diesem Buch sowohl unter Windows, macOS als auch Linux arbeiten. Alle nötigen Entwicklungswerkzeuge gibt es frei im Netz und werden gleich näher vorgestellt.

Kenntnisse in anderen Programmiersprachen sind zwar vorteilhaft, aber definitiv keine Grundvoraussetzung. Und falls es Dir doch einmal nicht schnell genug geht, kannst Du auch die Zusammenfassung am Ende jedes Kapitels lesen und dann immer noch entscheiden, ob Du das Kapitel Schritt für Schritt durcharbeiten möchtest.

1.2 Dieses Buch bricht mit einigen Konventionen!

Ich habe mich dazu entschlossen, mit einigen Konventionen zu brechen. So verzichte ich beispielsweise darauf, gleich zu Beginn auf Themen wie Objektorientierung oder Container einzugehen. Vielmehr lege ich Wert darauf, dass möglichst schnell erste Ergebnisse und Erfolgserlebnisse zu sehen sind. Das bedeutet natürlich nicht, dass auf sauberen Programmierstil oder fortgeschrittene Konzepte verzichtet wird. Allerdings sollten zuerst einige Grundlagen geschaffen werden, damit man weiterführende Themen versteht. Ich werde zugunsten der Lesbarkeit und der Lernkurve auch darauf verzichten, ein neues Thema sofort in all seinen tiefsten Details zu durchleuchten. Möchte jemand beispielsweise das Kochen erlernen, dann fängt er in der Regel mit einem einfachen Rezept an. Wenn nach fünfzehn oder zwanzig Minuten das erste unkomplizierte Essen auf dem Tisch steht, ist das eine tolle Motivation. Die physikalischen und chemischen Prozesse, die während des Kochvorgangs ablaufen, interessieren zunächst noch nicht. Später wird man vielleicht ausgefeilte Rezepte ausprobieren und sich dann ganz von selbst für die Details und Hintergründe interessieren. Hierzu ein kleiner Vergleich anhand eines Rezeptes, das auf unterschiedliche Weise präsentiert wird:

- **Variante 1:** Nimm einen schönen großen Topf, fülle ihn mit einem Liter Wasser und streue einen Teelöffel Salz hinein. Stelle den Topf auf den Herd, schalte die Platte auf Stufe 5 und warte, bis das Wasser kocht. Gib nun eine halbe Packung Nudeln hinzu, rühre gelegentlich um und warte, bis diese bissfest sind. Guten Appetit!
- **Variante 2:** Besorgen Sie sich einen temperaturbeständigen, oben offenen Behälter (sogenannter „Topf“), der gute Wärmeleiteigenschaften aufweist. Ideal ist das Element Aluminium oder auch eine Legierung, wie etwa Messing. Messen Sie anschließend mit einem geeigneten und möglichst gut geeichten Messbecher exakt einen Liter Dihydrogenmonoxid (ugs. „Wasser“) ab und füllen Sie dieses in den Behälter. Fügen Sie als Nächstes fünf Gramm Natriumchlorid (besser bekannt als „Salz“) hinzu. Dies dient ausschließlich geschmacklichen Zwecken und nicht etwa dem weit verbreiteten Irrtum, dass sich der Siedepunkt durch die Beigabe in einem relevanten Ausmaß ändert. Beachten Sie dabei jedoch, dass der Siedepunkt vom Luftdruck abhängig ist. Sollten Sie sich also in extremen Höhen oder Tiefen befinden, können Sie den Siedepunkt nach folgender Formel berechnen …

Auch ohne das Rezept in Variante zwei zu Ende zu führen sollte klar sein, worauf ich hinaus will: Details, Hintergründe und Exaktheit sind durchaus wichtig, aber in dieser Fülle für ein einfaches Beispiel nicht angebracht. Sie können einen erschlagen, die Motivation rauben und vom eigentlichen Geschehen ablenken. Das bedeutet selbstverständlich nicht, dass Themen grundsätzlich oberflächlich behandelt werden. Es geht vielmehr darum, Details auf einen geeigneteren Zeitpunkt zu verschieben.

1.3 Die Arbeit mit diesem Buch

Idealerweise arbeitest Du die Kapitel in der angegebenen Reihenfolge durch, denn sie bauen in der Regel auf vorangegangene Kenntnisse auf. Neue Themen werden zunächst erklärt und deren Anwendungsmöglichkeiten anhand von Beispielen verdeutlicht. Damit sich neu Gelerntes festigt, gibt es immer wieder kleine Übungsaufgaben. Sieh diese nicht als lästige Pflicht, sondern als Herausforderung an. Man lernt viel besser, wenn man sich Lösungen selbst erarbeitet, anstatt nur vorgefertigte Beispiele zu laden und zu starten. Natürlich werde ich Dir zu jeder Aufgabe Tipps und Hinweise geben. Im Anschluss gibt es dann eine Musterlösung, die Du mit Deiner eigenen Lösung vergleichen kannst.

Eine besondere Art von Aufgaben sind die sogenannten Fehlerquelltexte. Dabei handelt es sich um Beispiele, in die absichtlich Fehler eingebaut wurden. Diese Fehler führen dazu, dass sich das Programm entweder nicht ausführen lässt, oder dass es nicht so funktioniert wie erwartet. Selbst der erfahrenste Programmierer macht immer wieder Fehler und muss in der Lage sein, diese zu finden und zu korrigieren. Dennoch gibt es ganz bestimmte Stolpersteine, an denen man gerade anfangs immer wieder hängen bleibt. Die Fehlerquelltexte helfen dabei, ein Gespür für typische Fallen zu entwickeln und diese in Zukunft zu umgehen. Dies ist nützlich, um Fehler nicht nur schneller zu finden, sondern sie nach Möglichkeit auch von Anfang an so gut es geht zu vermeiden. Natürlich gibt es auch an dieser Stelle immer einige Tipps, wie man den Fehler eingrenzen, finden und beheben kann. Zudem wird auch eine korrigierte Fassung des Fehlerquelltextes gezeigt.

Abgesehen von den Vorschlägen wie Du am besten mit diesem Buch arbeitest, möchte ich Dir noch einige allgemeine Tipps mit auf den Weg geben. Wie so oft ist es bei neuen Dingen so, dass man anfangs große Fortschritte macht und dann plötzlich an eine Stelle gerät, an der es scheinbar nicht oder nur langsam vorangeht. Neue Themen wirken dann vielleicht übermäßig komplex oder deren Sinn erschließt sich nicht. Das kann manchmal ziemlich frustrierend sein, ist aber völlig normal. Am besten macht man in einer solchen Situation einfach eine längere Pause und beschäftigt sich mit etwas völlig anderem. Idealerweise mit etwas, das nichts mit Computern zu tun hat. Meistens kehrt man dann später mit einem anderen Blickwinkel zu dem Problem zurück und die Dinge scheinen wieder klarer. Auch wenn das vielleicht nach einem recht offensichtlichen Tipp klingt, solltest Du ihn dennoch beherzigen. Programmierung ist eine sehr fordernde Tätigkeit, die Pausen erfordert. „Eine Nacht darüber schlafen“ hat schon so manchem hartnäckigen Programmierfehler den Garaus gemacht und für frische Ideen gesorgt. Manchmal ist es auch hilfreich, sich einem anderen Thema zu widmen, wenn man an einer Stelle nicht weiterkommt. Blättere einfach einmal in vorherige Kapitel zurück oder schnuppere in neue Kapitel hinein.

Das Erlernen einer Programmiersprache (und das Programmieren an sich) kann man mit dem Erlernen eines Musikinstruments vergleichen: Man ist niemals fertig damit und hat nie ausgelernt. Selbst wenn man alle Features einer Programmiersprache kennt, stößt man immer wieder auf neue Programmiertechniken oder Themengebiete. Daher ist es wichtig, immer am Ball zu bleiben, um nicht den Anschluss zu verlieren.

Du solltest auch nicht vergessen, dass die Zeiten vorbei sind, in denen ein einsamer Programmierer im schwach beleuchteten Keller sitzt und völlig auf sich gestellt ein Programm entwickelt. Natürlich ist das immer noch möglich, aber nicht sonderlich sinnvoll. Program-

mierer sind längt keine winzige Randgruppe mehr, die ihr Wissen völlig neu erarbeitet und dann verschwörerisch hütet. In der heutigen Zeit beschäftigen sich sehr viele Menschen mit diesem Thema und sie tauschen sich über das Internet aus. Anstatt tagelang über einem Problem zu brüten, sucht man im Internet nach einer Lösung. Man kann fast immer davon ausgehen, dass jemand anderes zuvor genau das gleiche Problem hatte und es eine Lösung dafür gibt. Falls nicht, kann man immer noch selbst um Hilfe bitten. Das bedeutet natürlich nicht, dass man nicht selbst nachdenken sollte und sich alles nur noch zusammensuchen muss. Den eigenen Kopf einzuschalten ist die sinnvollste Lösung. Dennoch ist niemand mehr gezwungen, ein Problem auf sich alleine gestellt zu analysieren, ohne dass er Hilfe erhalten würde. Solltest Du auf offene Fragen anderer Programmierer stoßen, dann scheue Dich nicht davor, Deine Hilfe anzubieten. Du wirst staunen, wie schnell Du in der Lage sein wirst, Dein Wissen selbst wieder weiterzuvermitteln. Das ist natürlich auch nicht ganz uneigennützig, denn Du wirst selbst eine Menge lernen, wenn Du anderen hilfst. Wie viel Austausch Du betreiben möchtest, bleibt Dir natürlich selbst überlassen. Falls Du Dich dennoch dazu entschließt, im stillen Kämmerlein zu arbeiten, wirst Du eine Menge verpassen.

1.4 Das Kapitel zu Minecraft

Wie Dir sicher nicht entgangen ist, beschäftigt sich das letzte Kapitel in diesem Buch mit dem bekannten Spiel Minecraft. Wenn Du einen Raspberry Pi besitzt, kannst Du eine abgespeckte Version von Minecraft kostenfrei spielen und diese sogar mit Python programmieren. Das ist zwar kein Muss und nur optional, stellt aber eine tolle Möglichkeit dar, neu Gelerntes besser zu visualisieren. Statt Ergebnisse im Konsolenfenster auszugeben, kannst Du stattdessen eben Blöcke in einer Spielwelt platzieren. Es geht nicht darum, die kreativsten oder aufwendigsten Dinge zu bauen, sondern einfach Beispiele „zum Anfassen“ anzubieten. Du wirst an einigen Stellen im Buch Verweise auf passende Beispiele im Minecraft-Kapitel finden, mit denen Du experimentieren kannst.

1.5 Das Begleitmaterial zum Buch

Welche Entwicklungswerkzeuge Du für das Schreiben von Python-Programmen verwendest, kannst Du natürlich frei entscheiden. In diesem Buch wird die IDE (***I**ntegrated **D**evelopment **E**nvironment*, auf Deutsch *Integrierte Entwicklungsumgebung*) *PyCharm* von JetBrains verwendet. Dabei handelt es sich um ein mächtiges Werkzeug, das es in einer kostenfreien und einer kostenpflichtigen Version gibt. Die kostenfreie Version ist jedoch absolut ausreichend für alles, was in diesem Buch benötigt wird. Mehr zu PyCharm erfährst Du in Abschnitt 1.11.

Sämtliche im Buch abgedruckten Beispiele kannst Du unter *www.hanser-fachbuch.de* herunterladen. Gib dort oben rechts im Suchfeld ***Python 3*** ein, klicke auf das Lupen-Icon und anschließend auf den Buchtitel. Im Reiter **Extras** findest Du daraufhin den Link zu den Beispielen.

1.6 Anregungen? Kritik? Forum?

Das Schreiben dieses Buches hat mir viel Freude bereitet und ich hoffe, dass Du ebenso viel Spaß daran haben wirst, es zu lesen und damit zu arbeiten. Falls Du Anregungen, Verbesserungsvorschläge oder Kritik hast, dann schreibe mir eine E-Mail an *h.kalista@icloud.com*.

Ich habe während des gesamten Entstehungsprozesses dieses Buches darüber nachgedacht, ob ich eine Webseite mit einem Forum bereitstellen soll, in dem generelle Fragen zu Python gestellt werden können. Der Grund, warum ich darauf verzichtet habe, ist recht einfach: Es gibt bereits sehr viele Diskussionsforen und Hilfeseiten, auf denen sich eine große Nutzerschaft mit unterschiedlicher Erfahrung betätigt. Ein weiteres Forum ins Leben zu rufen, das zunächst mit Inhalt befüllt und eine Community generieren muss, ist da sicher nicht die effektivste Idee. Zudem gibt es in den Weiten des Internets wohl kaum eine Frage zu Python, die noch nicht gestellt und beantwortet wurde. Von daher möchte ich Dich dazu ermutigen, einfach einmal die vielen verschiedenen deutsch- und englischsprachigen Foren zu durchstöbern. Du wirst sehen, dass dort bereits eine große Community existiert.

Sollte es Neuigkeiten, Korrekturen oder Ergänzungen zum Buch geben, findest Du sie auf *www.hanser-fachbuch.de*. Gib im Suchfeld oben rechts ***Python 3*** ein, um zur Buchseite zu gelangen.

1.7 Die Geschichte von Python in 120 Wörtern

Ich weiß, dass Du sofort loslegen möchtest, daher mache ich es kurz: Python ist eine recht alte, aber immer noch aktuelle Sprache, die ständig weiterentwickelt wird. Die erste Version wurde zu Beginn der 1990er Jahre von Guido van Rossum entwickelt und erschien im Januar 1994 unter der Versionsnummer 1.0. Nach vielen kleineren Änderungen in den Versionen 1.1 und 1.2 wurde im Oktober 2000 die Version 2.0 veröffentlicht. Die wichtigste Neuerung war die sogenannte Garbage Collection (dabei handelt es sich vereinfacht gesagt um das automatische Aufräumen des Speichers). Ende 2008 wurde Version 3.0 veröffentlicht. Neben neuen Features wurden bestehende Sprachelemente grundlegend verändert und modernisiert, womit allerdings auch die Kompatibilität zu früheren Versionen verloren ging. Bis heute wird auch die Version 2.7 noch mit Updates versorgt und existiert parallel zur aktuellsten Version 3.7.

1.8 Was kann man mit Python machen (und was nicht)?

Häufig stellen sich gerade Programmiereinsteiger die Frage, welche die beste Programmiersprache sei. Das lässt sich genauso wenig beantworten wie die Frage, ob ein Hammer besser ist als ein Schraubenschlüssel. Beides sind Werkzeuge, nur eben für verschiedene Zwecke. Bei Programmiersprachen ist das ähnlich.

Python ist eine sehr vielseitige und flexible Sprache, die sich für fast alle Anwendungsgebiete eignet. Nicht ohne Grund nimmt die Verbreitung der Sprache gegenwärtig immer weiter zu. Zu den großen Stärken von Python zählen die leichte Erlernbarkeit, die gute Strukturierung der Sprache sowie deren Zugänglichkeit. Mit Python erhält man sehr schnell Ergebnisse, denn die Entwicklungszeiten sind gegenüber anderen Programmiersprachen sehr kurz. Nicht umsonst wird Python in der heutigen Zeit sehr oft als Einsteigersprache empfohlen. Das bedeutet jedoch nicht, dass sie nur für kleine Aufgaben geeignet ist. Ganz im Gegenteil: Sogar große und namhafte Unternehmen wie Google, die NASA, Dropbox und Facebook setzen auf die Vorteile von Python.

Möchtest Du ein kleines Tool schreiben, dass Dir bei der täglichen Arbeit am Rechner einige immer wiederkehrende Aufgaben abnimmt? Dann verwende Python! Hast Du vor, ein Programm mit grafischer Benutzeroberfläche zu entwickeln, ohne dafür übermäßig viel Aufwand betreiben zu müssen? Das ist mit Python möglich! Wolltest Du schon immer einmal deine eigene Heimautomatisierung realisieren und die Beleuchtung Deiner Zimmer, die Temperatur im Wohnzimmer oder die Rollos am Fenster per Knopfdruck steuern? Mit einem Raspberry Pi und Python kein Problem!

Python ist im wissenschaftlichen Bereich ebenso vertreten wie in der Anwendungsentwicklung und im Webbereich. Die Einsatzzwecke erstrecken sich dabei von kleinen, schnell zu erledigenden Aufgaben für zwischendurch bis hin zu großen, aufwendigen Projekten. Ein echter Allrounder eben.

Ein weiterer Vorteil von Python ist die Plattformunabhängigkeit. Es spielt keine Rolle, ob man für Windows, Linux oder macOS entwickeln möchte. Python ist für alle relevanten Systeme vorhanden und kostenfrei erhältlich. Auf vielen Unix-basierten Systemen ist Python sogar bereits vorinstalliert (Linux, macOS).

Allerdings gibt es auch einige Nachteile, denn es kann natürlich nicht „die“ perfekte Sprache für alles geben. Python eignet sich beispielsweise nicht für tiefgehende Systemprogrammierung. Niemand würde einen Treiber oder ein Betriebssystem in Python schreiben. Das liegt unter anderem daran, dass es sich um eine interpretierte Sprache handelt. Was genau das bedeutet, erfährst Du im nächsten Abschnitt. Zusammenfassend kann man jedoch sagen, dass ein Python-Programm nicht alleine lauffähig ist, sondern immer ein separates Programm zur Ausführung benötigt, den sogenannten Interpreter. Das bedeutet natürlich, dass man mit Python-Programmen in der Regel keine Wettrennen gewinnen wird. Wenn Ausführungsgeschwindigkeit der wichtigste Aspekt ist, ist es also nicht die erste Wahl. Zwar können rechenintensive Programmteile auch in anderen Sprachen entwickelt und dann eingebunden werden, aber „pure“ Python-Programme gehören nicht zu den Geschwindigkeitswundern. Es muss also je nach Anforderung abgewogen werden, was

wichtiger ist: schnelle Entwicklungszeit oder schnelle Ausführung. Eine eigene Grafik-Engine für moderne Computerspiele wird man mit Python also wohl nicht entwickeln.

Möchte man Apps für mobile Systeme wie Smartphones und Tablets entwickeln, so wird man zwangsläufig bei Objective-C oder Swift (iOS), respektive Java (Android) landen. Dies ist also einer der wenigen Bereiche, in denen man mit Python nicht weiterkommt.

Ein weiterer Anwendungsfall, bei dem Python den Kürzeren zieht, ist die Entwicklung im Embedded-Bereich. In diesem Gebiet ist jedes Byte wichtig und es wird an jeder Ecke optimiert. Einen Mikrocontroller wird man auch in den nächsten Jahren noch in C, C++ oder gar Assembler programmieren. Dennoch bleibt einem dieses Gebiet nicht gänzlich verschlossen, denn mit einem Raspberry Pi kann auch eine ganze Menge im Elektronikbereich realisiert werden.

1.9 Interpreter vs. Compiler

Bevor es gleich mit dem ersten Python-Programm losgeht, möchte ich noch auf einige Grundlagen von Programmiersprachen im Allgemeinen eingehen. Grundsätzlich ist es so, dass ein Computer zunächst weder mit einem in C, C++, oder Python verfassten Quelltext etwas anfangen kann. Wie bekannt sein dürfte, arbeitet ein Computer auf seiner elementarsten Ebene nur mit Nullen und Einsen – Strom an, Strom aus. Ein alter Hut, aber wichtig. Denn schließlich hat niemand von uns Lust, Programme in Nullen und Einsen in einen Computer zu hacken. Die nächsthöhere Ebene stellt die sogenannte Maschinen- oder Assemblersprache dar. Vereinfacht kann man sich das als einen kompakten Befehlssatz vorstellen, den ein Prozessor verstehen und verarbeiten kann. Dieser Befehlssatz besteht hauptsächlich aus simplen Rechen-, Vergleichs- und Sprunganweisungen. Im Vergleich zu Nullen und Einsen ist das zwar schon wesentlich bequemer, aber immer noch alles andere als lesbar. Du kannst Dich gerne davon überzeugen, indem Du Dir das folgende Beispiel anschaust:

```
movl   %r9d, %eax
movq   $0x0, -0xd0(%rbp)
movq   $0x0, -0x108(%rbp)
movq   -0x208(%rbp), %rcx
addq   $0x18, %rcx
movq   %rcx, %rdi
movq   %rax, %rdx
movq   %rax, %rcx
callq  0x10046f3f0
leaq   -0x120(%rbp), %rdi
movq   -0x208(%rbp), %rax
movq   0x18(%rax), %rcx
movq   %rcx, -0x220(%rbp)
```

Nicht gerade vielsagend, oder? Dennoch gibt es auch heute noch gewisse Situationen, in denen Maschinensprache weiterhin verwendet wird (direkter Zugriff auf die Hardware, zusätzliche Optimierung und so weiter). Und das wird sich so schnell auch nicht ändern. Zum Glück gibt es aber die sogenannten Hochsprachen, zu denen beispielsweise C, C++, C#, Basic und auch Python zählen. Diese Hochsprachen zeichnen sich durch ihre wesent-

lich bessere Lesbarkeit und ihren größeren Funktionsumfang aus. Wie Du Dir jetzt denken kannst, muss ein in einer solchen Hochsprache geschriebenes Programm zuerst in Maschinensprache umgewandelt werden, damit der Computer etwas damit anfangen kann. Wie dies geschieht, hängt von der verwendeten Programmiersprache ab. Grundsätzlich unterscheidet man zwischen kompilierten und interpretierten Sprachen. Bei kompilierten Sprachen wie etwa C oder C++ übersetzt ein sogenannter Compiler den Quellcode (also das in dieser Sprache geschriebene Programm) in Maschinencode und erzeugt somit ein ausführbares Programm, das von nun an von alleine lauffähig ist. Soll das Programm auf unterschiedlichen Plattformen (Windows, Linux, macOS) laufen, so muss es für die jeweiligen Plattformen separat kompiliert werden.

Bei interpretierten Sprachen wie beispielsweise Python läuft es etwas anders ab. Dreh- und Angelpunkt ist der sogenannte Python-Interpreter. Dieser verwandelt ein Python-Programm im ersten Schritt in sogenannten Bytecode. Dieser Bytecode ist für sich alleine noch nicht lauffähig, dafür ist dann die PVM *(**p**ython **v**irtual **m**achine)* zuständig. Wird das Python-Programm gestartet, führt diese virtuelle Maschine nacheinander alle Anweisungen aus, die im Bytecode stecken. Das Ganze ist unter der Haube ein recht komplexer Vorgang, aber glücklicherweise laufen diese Dinge automatisch im Hintergrund ab und wir müssen uns nicht darum kümmern. Dennoch halte ich es für wichtig, dass man mit den grundlegenden Konzepten vertraut ist.

Es sollte klar sein, warum eine interpretierte Sprache in der Regel langsamer ist als eine kompilierte und warum ein Anwender einen Python-Interpreter installiert haben muss, wenn er ein Python-Programm ausführen möchte.

1.10 Python 2.7 oder 3.7?

Wer sich mit Python beschäftigt, wird früher oder später mit der Frage konfrontiert, ob Python 2.7 oder Python 3.7 verwendet werden soll. Dies mag verwundern, da man eigentlich meinen sollte, dass immer die neueste Version vorzuziehen sei. Dass es nicht zwangsläufig so sein muss, zeigen die vielen Diskussionen zu diesem Thema im Internet. Die Verwirrung um diese beiden Versionen hängt mit der Tatsache zusammen, dass mit Version 3.0 einige einschneidende Änderungen vorgenommen wurden. Diese Änderungen waren so fundamental, dass die Abwärtskompatibilität zu älteren Python-Versionen verloren ging. Die neue Version beseitigte viele Unstimmigkeiten und wurde somit konsistenter und einsteigerfreundlicher. Zudem kamen neue Features hinzu und es wurde konsequent auf Unicode gesetzt (dies wurde auch in 2.x-Versionen unterstützt, musste aber separat markiert werden). Es versteht sich von selbst, dass weder Firmen noch Privatpersonen ihren gesamten Quellcode sofort auf Python 3.0 umstellen wollten oder konnten. Von daher wurde beschlossen, zunächst die Versionen 2.7 und 3.0 parallel laufen zu lassen und die ältere Version ebenfalls mit Updates und Fehlerkorrekturen zu versorgen. Wer wollte, konnte neue Projekte mit Version 3.0 erstellen und bei alten Projekten dennoch vom Support für Version 2.7 profitieren. Es wurden sogar einige Features von neueren Python-Versionen in Version 2.7 übernommen, solange diese nicht die Kompatibilität gefährdeten. Python 2.7 sollte ursprünglich bis 2015 gepflegt werden, dieses Datum wurde jedoch bis 2020 verlän-

gert. Doch was bedeutet das nun? Soll man mit der aktuellsten Version von Python beginnen, oder doch lieber erst mit 2.7?

Zunächst einmal sollte man sich bewusst sein, dass es sich bei den beiden Versionen nicht um völlig verschiedene Sprachen handelt. Man muss also nicht von Grund auf neu lernen, sondern sich nur mit einigen Änderungen auseinandersetzen. Der Lernerfolg sollte somit also identisch sein. Dennoch möchte ich Dir empfehlen, mit Python 3.7 einzusteigen und Dich bei Bedarf mit den Unterschieden zu Version 2.7 vertraut zu machen.

Etwas genauer hinschauen musst Du allerdings, wenn Du neben dem reinen Lernen der Sprache auf ein bestimmtes Ziel hinarbeitest. In Python kann man sogenannte Libraries (Bibliotheken) einbinden, die bestimmte Funktionalitäten bieten. Dies können beispielsweise Bibliotheken für Netzwerk-, Grafik-, oder KI-Programmierung sein. Solche Bibliotheken sind ebenfalls in Python geschrieben und stammen meist aus der Open-Source-Szene. Benötigt man unbedingt eine bestimmte Bibliothek, die es allerdings nur in Version 2.7 gibt, hat man nur zwei Möglichkeiten: Man verwendet für das eigene Projekt ebenfalls Python 2.7 oder man versucht, die Bibliothek umzuschreiben.

1.11 Die Entwicklungsumgebung PyCharm

Wir wissen jetzt, was man mit Python so alles anstellen kann, wie es unter der Haube funktioniert und welche Version wir benötigen. Doch womit schreibt man am besten Python-Programme? Wie so oft lässt sich diese Frage nicht pauschal beantworten. Im Grunde genügt ein einfacher Texteditor und ein installierter Python-Interpreter. Allerdings leben wir im Jahre 2017 und es spricht nichts gegen etwas Komfort. Für die meisten Programmiersprachen gibt es sowohl einfache Tools, als auch sehr leistungsfähige Entwicklungsumgebungen, die einem das Leben als Programmierer erleichtern. Bei letzteren spricht man von sogenannten IDEs. Das ist die Abkürzung für den englischen Begriff ***I**ntegrated **D**evelopment **E**nvironment*, zu Deutsch *Integrierte Entwicklungsumgebung*. Eine solche IDE besteht aus mehreren Komponenten. Neben einem Texteditor mit vielen auf die jeweilige Programmiersprache zugeschnittenen Bequemlichkeitsfunktionen findet man meistens auch einen Debugger (ein Tool, mit dem man Programmfehler aufspüren kann), eine Quellcodeverwaltung und Werkzeuge für Analyse und Tests. Der Vorteil solcher IDEs besteht darin, dass man alles Nötige zentral in einem Programm gebündelt hat (wie der Name durchaus erahnen lässt).

Einige dieser Tools sind kostenlos erhältlich, andere sind nur für viel Geld zu haben oder stark in ihrer Funktion eingeschränkt. In diesem Buch wird die Community Edition von PyCharm verwendet. Diese ist für Windows, macOS und Linux kostenlos verfügbar, hat alle Features, die wir benötigen, und sie darf sogar für kommerzielle Projekte verwendet werden. PyCharm ist ein sehr mächtiges Werkzeug, das eine entsprechende Einarbeitung erfordert. Allerdings werde ich darauf verzichten, gleich zu Beginn alle Features zu erklären. Vielmehr beschränke ich mich darauf, immer nur die Teile anzusprechen, die für das aktuelle Thema auch wirklich wichtig sind. Es steht Dir natürlich frei, eine andere IDE zu benutzen.

1.11.1 Alternativen zu PyCharm

Neben dem gerade erwähnten PyCharm und der Verwendung eines schlichten Texteditors gibt es selbstverständlich noch andere Entwicklungsumgebungen, von denen ich Dir hier drei vorstellen möchte:

IDLE

Ein bekanntes, schlicht gehaltenes und im Grunde völlig ausreichendes Programm ist *IDLE*, das in der Regel zusammen mit dem Python-Interpreter installiert wird. Wenn Du nur mal eben eine kleine Aufgabe mit Python erledigen willst, dann ist *IDLE* sicherlich nicht verkehrt (es verfügt sogar über einen Debugger). Man muss jedoch fairerweise dazu sagen, dass es sich bei diesem Programm nicht unbedingt um das modernste und benutzerfreundlichste handelt. Starte es einfach einmal und probiere aus, ob es Dir liegt. Wenn Du Ubuntu 18.04 verwendest, musst Du es noch per `sudo apt-get install idle3` installieren. Unter Windows und macOS wird es zusammen mit dem Python-Interpreter ausgeliefert.

Thonny

Mehr Komfort, eine modernere Benutzeroberfläche und zusätzliche Features bietet *Thonny*, das Du unter *http://thonny.org* für Windows und macOS kostenfrei herunterladen kannst. Für die Installation unter Linux findest Du unter dem genannten Link alle nötigen Informationen. In Kapitel 13 (Minecraft) wird *Thonny* für die Python-Entwicklung verwendet, Du wirst es also noch näher kennenlernen.

Visual Studio Code

Sehr viele Features und Möglichkeiten bietet Dir *Visual Studio Code*, das Du unter *https://code.visualstudio.com* ebenfalls kostenfrei für Windows, macOS und Linux herunterladen kannst. Wenn Du Ubuntu 18.04 verwendest, findest Du es auch direkt unter **Ubuntu Software**. *Visual Studio Code* ist ein sehr mächtiges Werkzeug, das für die Entwicklung mit den unterschiedlichsten Programmiersprachen eingesetzt wird. Um es zusammen mit Python zu verwenden, sind jedoch einige vorbereitende Schritte notwendig. Diese sind übersichtlich auf der Seite *https://code.visualstudio.com/docs/python/python-tutorial* beschrieben.

1.12 Python-Interpreter installieren

Python ist eine interpretierte Sprache und benötigt somit einen entsprechenden Interpreter. Diesen gibt es kostenlos für fast alle relevanten Betriebssysteme und bei vielen gehört er bereits zum Lieferumfang (beispielsweise macOS und so gut wie alle Linux-Distributionen). Im Folgenden gehe ich kurz auf die Installation unter Windows, macOS und Linux (Ubuntu 18.04) ein. Falls Du eine andere Linux-Distribution verwendest, kann sich die Vorgehensweise der Installation eventuell unterscheiden.

HINWEIS: Python 3.7 ist nicht abwärtskompatibel zu Python 2.7. Wenn Du also auch mit Python 2.7 arbeiten möchtest, dann musst Du beide Interpreter installieren.

1.12.1 Python-Interpreter unter Windows installieren

Windows-Betriebssysteme enthalten leider keinen vorinstallierten Python-Interpreter, weder den der Version 2.7 noch den der Version 3.7. Um diesen zu installieren, gehe zunächst auf die Seite *https://www.python.org/downloads* und klicke auf den Button **Download Python 3.7.0** um die Installationsdatei herunterzuladen. Zusätzlich hast Du noch die Möglichkeit, weiter unten auf der Seite verschiedene Python-Installationen herunterzuladen. Diese unterscheiden sich anhand der Version, der Architektur (32 oder 64 Bit) und der Art des Installationsprogramms. Das ist an dieser Stelle allerdings nicht von Bedeutung und Du kannst beruhigt die Standardversion verwenden.

HINWEIS: Python ab Version 3.5 benötigt mindestens Windows Vista oder neuer. Solltest Du noch Windows XP verwenden, dann musst Du auf Python 3.4 ausweichen.

Folge nun dem Installationsverlauf und wähle die Option **Add Python 3.7 to PATH**. Das sorgt dafür, dass Du Python aus der Eingabeaufforderung (Kommandozeile) von überall her aufrufen kannst, ohne vorher extra in ein Verzeichnis wechseln zu müssen.

Um zu prüfen, ob Python korrekt installiert wurde, hast Du zwei Möglichkeiten. Öffne zuerst die Eingabeaufforderung und gib wahlweise `python` oder `py` ein. Wenn alles geklappt hat, solltest Du folgende Bildschirmausgabe sehen:

```
Microsoft Windows [Version 10.0.16299.431]
(c) 2017 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Heiko>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Der Python-Interpreter wird gestartet, gibt die aktuelle Version aus und zeigt mit `>>>` an, dass er bereit ist, Python-Befehle zu empfangen. Du kannst das Fenster jetzt entweder einfach schließen, oder mit der Eingabe von `exit()` und der **Enter**-Taste den Interpreter beenden.

Die zweite Möglichkeit besteht darin, über das Startmenü den Eintrag **Python 3.7 -> Python 3.7 (32 Bit)** anzuwählen. In diesem Fall öffnet sich ebenfalls die Eingabeaufforderung. Dieses Mal allerdings direkt mit gestartetem Python-Interpreter. Weiterhin findest Du im Startmenü noch Einträge zur Dokumentation und der mitgelieferten Entwicklungsumgebung *IDLE*. Diese wurde weiter oben bereits als Alternative zu PyCharm angesprochen. Wir