

Der Weg zum Python-Profi

Ein Best-Practice-Buch für sauberes Programmieren

Al Sweigart



dpunkt.verlag

Papier
plus⁺
PDF.

Zu diesem Buch – sowie zu vielen weiteren dpunkt.büchern – können Sie auch das entsprechende E-Book im PDF-Format herunterladen. Werden Sie dazu einfach Mitglied bei dpunkt.plus⁺:

www.dpunkt.plus

Al Sweigart

Der Weg zum Python-Profi

Ein Best-Practice-Buch für sauberes Programmieren



dpunkt.verlag

Al Sweigart

Lektorat: Gabriel Neumann

Lektoratsassistentz: Anja Weimer

Übersetzung: Volkmar Gronau, G&U Language & Publishing Services GmbH, Flensburg, www.GundU.com

Copy-Editing: Claudia Lötschert, www.richtiger-text.de

Satz: Ulrich Borstelmann, www.borstelmann.de

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 978-3-86490-874-3

PDF 978-3-96910-677-8

ePub 978-3-96910-678-5

mobi 978-3-96910-679-2

1. Auflage 2022

Translation Copyright für die deutschsprachige Ausgabe © 2022 dpunkt.verlag GmbH

Wieblingen Weg 17

69123 Heidelberg

Copyright © 2021 by Al Sweigart. Title of English-language original: *Beyond the Basic Stuff with Python: Best Practices for Writing Clean Code*, ISBN 978-1-59327-966-0,

published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103.

The German-language edition Copyright © 2022 by dpunkt.verlag under license by No Starch Press Inc.

All rights reserved.

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.

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

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

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

5 4 3 2 1 0

Inhalt

Der Autor	xvi
Der Fachgutachter	xvi
Danksagung	xvii
Einleitung	1
Wer dieses Buch lesen sollte und warum	2
Über dieses Buch	3
Ihr Weg zur Programmierung	5
1 Fehlermeldungen und Recherche	7
Python-Fehlermeldungen verstehen	8
Tracebacks untersuchen	8
Fehlermeldungen recherchieren	11
Fehler vermeiden mit Lintern	13
Um Hilfe bitten	14
Geben Sie gleich ausreichend Informationen, um Rückfragen zu vermeiden	15
Formulieren Sie Ihre Fragen als Fragen	15
Stellen Sie Ihre Fragen auf einer geeigneten Website	16
Geben Sie das Problem in der Überschrift an	16
Erklären Sie, was der Code tun soll	16
Geben Sie die vollständige Fehlermeldung an	17
Teilen Sie Ihren Code vollständig mit	17
Gestalten Sie Ihren Code durch saubere Formatierung lesbar	18
Beschreiben Sie, was Sie bereits versucht haben	19
Beschreiben Sie Ihre Ausstattung	19
Ein Beispiel für eine gut gestellte Frage	20
Zusammenfassung	21

2 Die Einrichtung der Umgebung und die Befehlszeile	23
Das Dateisystem	24
Pfade in Python	25
Das Benutzerverzeichnis	25
Das aktuelle Arbeitsverzeichnis	26
Absolute und relative Pfade	27
Programme und Prozesse	28
Die Befehlszeile	29
Ein Terminalfenster öffnen	30
Programme an der Befehlszeile ausführen	31
Befehlszeilenargumente	32
Python-Code mit <code>-c</code> an der Befehlszeile ausführen	33
Python-Programme an der Befehlszeile ausführen	34
Py.exe	34
Befehle aus einem Python-Programm heraus ausführen	35
Tipparbeit durch Tabulatorvervollständigung sparen	35
Der Befehlsverlauf	36
Gebräuchliche Befehle	36
PATH und andere Umgebungsvariablen	44
Umgebungsvariablen anzeigen	44
Die Umgebungsvariable PATH	45
Die Umgebungsvariable PATH in der Befehlszeile ändern	46
Ordner in Windows dauerhaft zu PATH hinzufügen	47
Ordner in macOS und Linux dauerhaft zu PATH hinzufügen	49
Python-Programme außerhalb der Befehlszeile ausführen	49
Python-Programme in Windows ausführen	49
Python-Programme in macOS ausführen	51
Python-Programme in Ubuntu Linux ausführen	51
Zusammenfassung	52
3 Codeformatierung mit Black	53
Wie man Freunde und Kollegen vergrault	54
PEP 8 und andere Stilrichtlinien	54
Horizontale Abstände	55
Leerzeichen zur Einrückung verwenden	55
Abstände innerhalb einer Zeile	57
Vertikale Abstände	60
Beispiel für vertikale Abstände	60
Empfohlene Vorgehensweisen für vertikale Abstände	62

Black: Der kompromisslose Codeformatierer	63
Black installieren	63
Black an der Befehlszeile ausführen	64
Black für einzelne Abschnitte Ihres Codes ausschalten	67
Zusammenfassung	68
4 Aussagekräftige Namen	69
Groß- und Kleinschreibung	70
Namenskonventionen in PEP 8	71
Namen geeigneter Länge	72
Zu kurze Namen	72
Zu lange Namen	73
Leicht zu findende Namen	75
Scherze, Wortspiele und Anspielungen vermeiden	75
Integrierte Namen nicht überschreiben	76
Die allerschlechtesten Variablennamen	78
Zusammenfassung	78
5 Codegerüche	79
Duplizierter Code	80
Magische Zahlen	82
Auskommentierter und toter Code	84
Print-Debugging	86
Variablen mit numerischen Suffixen	87
Klassen statt Funktionen oder Module	88
Verschachtelte Listennotation	89
Leere except-Blöcke und nichtssagende Fehlermeldungen	91
Legenden über Code Smells	92
Legende: Funktionen sollten nur eine return-Anweisung am Ende aufweisen	93
Legende: Funktionen mit einer try-Anweisung dürfen keine anderen Anweisungen enthalten	93
Legende: Flag-Argumente sind schlecht	94
Legende: Globale Variablen sind schlecht	95
Legende: Kommentare sind unnötig	96
Zusammenfassung	97

6 Pythonischer Code	99
Python-Zen	100
Aussagekräftige Einrückungen	104
Leistung mit dem Modul <code>timeit</code> messen	105
Häufig falsch angewendete Syntax	108
Verwenden Sie <code>enumerate()</code> statt <code>range()</code>	108
Verwenden Sie <code>with</code> statt <code>open()</code> und <code>close()</code>	109
Verwenden Sie <code>is</code> statt <code>==</code> zum Vergleich mit <code>None</code>	110
Stringformatierung	111
Verwenden Sie Rohstrings bei einer großen Anzahl von Backslashes ...	111
F-Strings	112
Flache Kopien von Listen	113
Pythonischer Umgang mit Dictionarys	114
Verwenden Sie <code>get()</code> und <code>setdefault()</code> für Dictionarys	114
Verwenden Sie <code>collections.defaultdict</code> für Standardwerte	116
Verwenden Sie Dictionarys statt <code>switch</code> -Konstruktionen	117
Bedingte Ausdrücke: Pythons »hässlicher« ternärer Operator	118
Variablenwerte	120
Zuweisungs- und Vergleichsoperatoren verketteten	120
Eine Variable auf Gleichheit mit mehreren Werten prüfen	121
Zusammenfassung	121
7 Programmierjargon	123
Definitionen	124
Die Sprache Python und der Interpreter Python	124
Garbage Collection	125
Literele	126
Schlüsselwörter	126
Objekte, Werte und Identitäten	127
Elemente	131
Veränderbare und unveränderbare Objekte	131
Indizes, Schlüssel und Hashes	134
Container-, Folgen-, Maps- und Set-Datentypen	137
Dunder- oder magische Methoden	138
Module und Pakete	139
Aufrufbare Objekte und Objekte erster Klasse	139

Häufig verwechselte Begriffe	141
Anweisungen und Ausdrücke	141
Blöcke, Klauseln und Rümpfe	142
Variablen und Attribute	143
Funktionen und Methoden	143
Iterierbare Objekte und Iteratoren	144
Syntax-, Laufzeit- und semantische Fehler	146
Parameter und Argumente	148
Implizite und explizite Typumwandlung	148
Eigenschaften und Attribute	149
Bytecode und Maschinencode	149
Skripte und Programme, Skriptsprachen und Programmiersprachen	150
Bibliotheken, Frameworks, SDKs, Engines und APIs	150
Zusammenfassung	151
Literatur	152
8 Häufige Fallstricke in Python	153
Elemente zu Listen hinzufügen und entfernen	154
Veränderbare Werte kopieren	161
Standardargumente	164
Strings zusammenbauen	166
Sortierung mit sort()	168
Genauigkeit von Fließkommazahlen	169
Verkettung von Ungleichheitsoperatoren	172
Das Komma in einelementigen Tupeln	172
Zusammenfassung	173
9 Exotische Eigenarten von Python	175
256 ist 256, aber 257 ist nicht 257	175
String-Interning	177
Die Bedeutung von ++ und -- in Python	178
Alles von nichts	179
Boolesche Werte als Integer	180
Verkettung unterschiedlicher Operatoren	182
Schwereelosigkeit in Python	182
Zusammenfassung	183

10 Zweckmäßige Funktionen	185
Funktionsnamen	186
Der Umfang von Funktionen	186
Funktionsparameter und -argumente	189
Standardargumente	190
Argumente mit * und ** an Funktionen übergeben	190
Variadische Funktionen mit * erstellen	192
Variadische Funktionen mit ** erstellen	195
Wrapper-Funktionen mit * und ** erstellen	197
Funktionale Programmierung	198
Nebenwirkungen	198
Funktionen höherer Ordnung	200
Lambda-Funktionen	201
Zuordnung und Filterung mit Listennotation	202
Der Datentyp von Rückgabewert	203
Ausnahmen auslösen oder Fehlercodes zurückgeben	206
Zusammenfassung	207
11 Kommentare, Docstrings und Typhinweise	209
Kommentare	210
Formatierung von Kommentaren	211
Inline-Kommentare	212
Erklärende Kommentare	213
Kommentare zur Gliederung	213
»Lessons-Learned-Kommentare«	214
Rechtliche Hinweise	215
Professionelle Formulierung	215
Codetags und TODO-Kommentare	216
Magische Kommentare und Quelldateicodierung	217
Docstrings	217
Typhinweise	220
Tools zur statischen Analyse	222
Typhinweise für mehrere Typen	225
Typhinweise für Listen, Dictionarys u. Ä.	226
Rückportierung von Typhinweisen mithilfe von Kommentaren	227
Zusammenfassung	228

12 Versionssteuerung mit Git	231
Commits und Repositories in Git	232
Neue Python-Projekte mit Cookiecutter erstellen	232
Git installieren	235
Git-Benutzername und E-Mail-Adresse angeben	236
GUI-Werkzeuge für Git installieren	236
Der Arbeitsablauf in Git	237
Der Dateistatus in Git	237
Wozu gibt es bereitgestellte Dateien?	239
Ein Git-Repository erstellen	240
Zu verfolgende Dateien hinzufügen	241
Einzelne Dateien ignorieren	243
Änderungen mit Commit bestätigen	244
Änderungen mit git diff vor dem Commit einsehen	245
Änderungen mit git difftool in einer GUI-Anwendung einsehen	247
Häufigkeit von Commits	248
Dateien löschen	249
Dateien umbenennen und verschieben	250
Das Commitprotokoll einsehen	252
Frühere Versionen wiederherstellen	253
Unbestätigte lokale Änderungen rückgängig machen	253
Bereitstellung einer Datei aufheben	254
Die letzten Commits zurücknehmen	254
Zurücksetzen einer einzelnen Datei zu einem bestimmten Commit	255
Den Commitverlauf ändern	256
GitHub und git push	257
Ein bestehendes Repository auf GitHub übertragen	258
Ein GitHub-Repository klonen	259
Zusammenfassung	260
13 Leistungsmessung und Algorithmusanalyse	261
Das Modul timeit	262
Der Profiler cProfile	265
Komplexitätsanalyse	267
Ordnungen	268
Ein Bücherregal als Metapher für Ordnungen	269
Worst Case und Best Case	273

Die Ordnung Ihres Codes bestimmen	275
Warum Terme niedriger Ordnungen und Koeffizienten keine Rolle spielen	276
Beispiele für die Komplexitätsanalyse	278
Die Ordnung gängiger Funktionsaufrufe	281
Komplexitätsanalyse im Überblick	282
Die Ordnung spielt bei kleinem n keine Rolle – und n ist gewöhnlich klein	284
Zusammenfassung	284
14 Praxisprojekte	287
Turm von Hanoi	288
Die Ausgabe	289
Der Quellcode	290
Den Code schreiben	292
Vier gewinnt	301
Die Ausgabe	301
Der Quellcode	302
Den Code schreiben	306
Zusammenfassung	315
15 Klassen	317
Formulare als Veranschaulichung	318
Objekte aus Klassen erstellen	320
Eine einfache Klasse erstellen: WizCoin	321
Methoden, <code>__init__()</code> und der Parameter <code>self</code>	324
Attribute	325
Private Attribute und private Methoden	326
Die Funktion <code>type()</code> und das Attribut <code>__qualname__</code>	328
OOP- und Nicht-OOP-Code im Vergleich	329
Klassen für reale Objekte	335
Zusammenfassung	336

16 Vererbung	339
Wie Vererbung funktioniert	340
Methoden überschreiben	342
Die Funktion super()	344
Komposition statt Vererbung	346
Nachteile der Vererbung	348
Die Funktionen isinstance() und subclass()	350
Klassenmethoden	351
Klassenattribute	354
Statische Methoden	354
Wann brauchen Sie Klassenmerkmale und statische Methoden?	355
Schlagwörter der objektorientierten Programmierung	355
Kapselung	356
Polymorphismus	356
Wann Sie die Vererbung nicht nutzen sollten	357
Mehrfachvererbung	358
Die Reihenfolge der Methodenauflösung	360
Zusammenfassung	362
17 Pythonische OOP: Eigenschaften und Dunder-Methoden	363
Eigenschaften	364
Attribute in Eigenschaften umwandeln	364
Set-Methoden zur Datenvalidierung	367
Schreibgeschützte Eigenschaften	369
Wann Sie Eigenschaften verwenden sollten	371
Dunder-Methoden	371
Dunder-Methoden zur Stringdarstellung	372
Numerische Dunder-Methoden	375
Reflektierte numerische Dunder-Methoden	379
Direkte Dunder-Methoden für erweiterte Zuweisungsoperatoren	381
Dunder-Methoden für Vergleiche	383
Zusammenfassung	388
Stichwortverzeichnis	391

Für meinen Neffen Jack

Der Autor

Al Sweigart ist Softwareentwickler und Fachbuchautor und lebt in Seattle. Python ist seine bevorzugte Programmiersprache, für die er bereits mehrere Open-Source-Module entwickelt hat. Er hat mehrere Programmierbücher für Einsteiger geschrieben, darunter *Routineaufgaben mit Python automatisieren* und *Cooler Spiele mit Scratch 3*, die ebenfalls beim dpunkt.verlag erschienen sind. Die englische Originalversion seiner Bücher steht unter einer Creative-Commons-Lizenz kostenlos auf seiner Website <https://www.inventwithpython.com/> zur Verfügung. Seine Katze Zophie wiegt 11 Pfund.

Der Fachgutachter

Kenneth Love ist Programmierer und Lehrer und richtet Konferenzen aus. Er ist Mitarbeiter von Django und ein PSF Fellow. Zurzeit arbeitet er als technischer Leiter und Softwareingenieur für O'Reilly Media.

Danksagung



Es ist irreführend, dass nur mein Name auf dem Titelbild steht, denn ohne die Bemühungen vieler anderer Personen hätte es dieses Buch nie gegeben. Ich möchte meinem Herausgeber Bill Pollock, meinen Lektoren Frances Saux, Annie Choi, Meg Sneeringer und Jan Cash, Herstellungsleiterin Maureen Forys, Korrektorin Anne Marie Walker und Chefredakteurin Barbara Yien danken. Ein weiteres Dankeschön geht an Josh Ellingson für ein weiteres hervorragendes Titelbild, an meinen Fachgutachter Kenneth Love und an all die großartigen Freunde, die ich in der Python-Community kennengelernt habe.

Einleitung



Hello again, world! In den späten 90er-Jahren habe ich als programmierender Teenager und Möchtegernhacker immer die neueste Ausgabe von *2600: The Hacker Quarterly* studiert. Eines Tages fand ich endlich den Mut, an dem von der Zeitschrift organisierten monatlichen Treffen in meiner Heimatstadt teilzunehmen, und war beeindruckt davon, wie viel diese Leute alle wussten! (Später erkannte ich jedoch, dass viele von ihnen über weit mehr Selbstvertrauen als echte Kenntnisse verfügten.) Das ganze Treffen über lauschte ich ehrfürchtig nickend dem, was die anderen zu sagen hatten, und versuchte, den Unterhaltungen zu folgen. Danach war ich entschlossen, jede Gelegenheit zu nutzen, um mehr über Computer, Programmierung und Netzwerksicherheit zu lernen, sodass ich mich beim nächsten Treffen an den Gesprächen beteiligen konnte.

Beim nächsten Treffen hörte ich jedoch weiterhin nur zu und fühlte mich im Vergleich zu allen anderen minderbemittelt. Also fasste ich erneut den Vorsatz, zu lernen und klug genug zu werden, um mitzuhaltten. Ich vertiefte mein Wissen Monat für Monat, hatte aber immer das Gefühl, hinterherzuhinken. Schließlich er-

kannte ich, wie umfangreich das Gebiet der Informatik war, und befürchtete, niemals genug wissen zu können.

Ich wusste damals zwar schon mehr als meine Schulfreunde, doch meine Kenntnisse reichten mit Sicherheit nicht aus, um als Softwareentwickler arbeiten zu können. In den 90er-Jahren gab es Google, YouTube und Wikipedia noch nicht, aber ich hätte auch gar nicht gewusst, wie ich sie hätte nutzen sollen. Es war mir nie klar, womit ich mich als Nächstes befassen sollte. Stattdessen lernte ich, Hallo-Welt-Programme in verschiedenen Programmiersprachen zu schreiben. Dabei hatte ich jedoch nie das Gefühl, echte Fortschritte zu machen. Ich wusste nicht, wie ich jemals über die Grundlagen hinauskommen sollte.

Zur Softwareentwicklung gehört weit mehr als Schleifen und Funktionen. Wenn Sie einen Anfängerkurs absolviert oder ein Programmierbuch für Einsteiger gelesen haben und sich nach weiterführenden Informationen umsehen, landen Sie aber leider meistens nur bei weiteren Hallo-Welt-Tutorials. Programmierer nennen diese Phase die *Wüste der Verzweiflung*: Sie bewegen sich ziellos durch unterschiedliche Lernstoffe, ohne das Gefühl zu haben, Fortschritte zu machen. Für Material, das sich an Anfänger richtet, wissen Sie schon zu viel, aber es fehlt Ihnen an Erfahrung, um sich den komplizierteren Themen zu widmen.

Wenn Sie sich in dieser Wüste befinden, kommen Sie sich wie ein Hochstapler vor. Sie haben nicht das Gefühl, ein »richtiger« Programmierer zu sein oder Code so schreiben zu können, wie »richtige« Programmierer es tun. Dieses Buch habe ich für Personen geschrieben, denen es genau so geht. Wenn Sie die Grundlagen von Python kennen, hilft es Ihnen, ein besserer Softwareentwickler zu werden und dieses Gefühl der Verzweiflung zu überwinden.

Wer dieses Buch lesen sollte und warum

Dieses Buch richtet sich an Leser, die bereits einen Grundkurs in Python absolviert haben und mehr lernen möchten. Bei diesem Grundkurs kann es sich um mein Buch *Routineaufgaben mit Python automatisieren* (dpunkt.verlag, 2020), das Buch *Python 3 Crashkurs* von Eric Matthes (dpunkt.verlag, 2020) oder auch eine Onlineschulung handeln.

Solche Einführungen können Ihre Begeisterung für die Programmierung wecken, allerdings gibt es danach immer noch viel zu lernen. Wenn Sie das Gefühl haben, noch nicht das Niveau eines professionellen Programmierers erreicht zu haben, und nicht wissen, wie Sie dorthin gelangen sollen, ist dies das richtige Buch für Sie.

Vielleicht haben Sie ja auch in einer anderen Sprache zu programmieren gelernt und möchten nun unmittelbar in das Python-Umfeld mit allen seinen Tools wech-

seln, ohne die typischen Hello-World-Grundlagen wiederzukäuen. In diesem Fall brauchen Sie nicht erst Hunderte von Seiten durcharbeiten, die die grundlegende Syntax erklären. Es reicht, wenn Sie sich den Artikel »Learn Python in Y Minutes« auf <https://learnxinyminutes.com/docs/python> ansehen oder Eric Matthes' Python-Spickzettel von »Python Crash Course – Cheat Sheet« auf <https://ehmatthes.github.io/pcc/cheatsheets/README.html> herunterladen, bevor Sie dieses Buch in Angriff nehmen.

Über dieses Buch

In diesem Buch geht es nicht nur um Python-Syntax für Fortgeschrittene, sondern auch um die Verwendung der Befehlszeile und von Tools wie Codeformatierern, Lintern und Versionssteuerung, die auch professionelle Entwickler einsetzen.

Ich erkläre Ihnen, was Code gut lesbar macht und wie Sie sauberen Code schreiben können. Zur Veranschaulichung dieser Prinzipien stelle ich einige Programmierprojekte vor. Dies soll zwar kein Lehrbuch in Informatik werden, aber dennoch werden wir uns auch mit der O-Notation und objektorientierter Entwicklung beschäftigen.

Ein Buch allein reicht nicht aus, um jemanden zu einem professionellen Softwareentwickler zu machen. Ich habe dieses Buch geschrieben, um Ihre Kenntnisse zu vertiefen und Ihnen dadurch auf diesem Weg weiterzuhelfen. Dabei spreche ich auch einige Themen an, die Sie anderenfalls nur nach und nach durch Erfahrung lernen würden. Dieses Buch verschafft Ihnen eine solide Grundlage, sodass Sie gut für neue Herausforderungen gerüstet sind.

Ich rate Ihnen zwar dazu, das Buch von vorn bis hinten zu lesen, aber wenn ein Thema Sie besonders interessiert, können Sie auch gern zu dem betreffenden Kapitel vorblättern.

Teil I: Erste Schritte

- **Kapitel 1: Fehlermeldungen und Recherche** Zeigt Ihnen, wie Sie erfolgreich Fragen stellen und selbstständig Lösungen finden können. Außerdem erfahren Sie hier, wie Fehlermeldungen zu lesen sind und welche Verhaltensregeln Sie beachten müssen, wenn Sie online um Hilfe bitten.
- **Kapitel 2: Die Einrichtung der Umgebung und die Befehlszeile** Erklärt, wie Sie sich an der Befehlszeile zurechtfinden und wie Sie Ihre Entwicklungsumgebung sowie die Umgebungsvariable PATH einrichten.

Teil II: Werkzeuge, Techniken und bewährte Vorgehensweisen

- **Kapitel 3: Codeformatierung mit Black** Beschreibt die Stilrichtlinie PEP 8 und erklärt, wie Sie Ihren Code formatieren sollten, um ihn besser lesbar zu machen. Sie erfahren hier auch, wie Sie diesen Vorgang mit dem Codeformatierungswerkzeug Black automatisieren können.
- **Kapitel 4: Aussagekräftige Namen** Erklärt, wie Sie Variablen und Funktionen benennen sollten, um Ihren Code übersichtlicher zu gestalten.
- **Kapitel 5: Codegerüche** Beschreibt Warnzeichen, die auf mögliche Bugs in Ihrem Code hinweisen.
- **Kapitel 6: Pythonischer Code** Erklärt, was Code *pythonisch* macht, und zeigt Vorgehensweisen auf, um solchen idiomatischen Python-Code zu schreiben.
- **Kapitel 7: Programmierjargon** Erklärt Fachbegriffe, die in der Programmierung verwendet werden, sowie Begriffe, die oft verwechselt werden.
- **Kapitel 8: Häufige Fallstricke in Python** Beschreibt Aspekte von Python, die oft zu Missverständnissen oder zu Bugs führen, und zeigt Korrekturmaßnahmen sowie Möglichkeiten auf, solche Schwierigkeiten zu vermeiden.
- **Kapitel 9: Exotische Eigenarten von Python** Behandelt einige seltsame Eigenheiten von Python wie String-Interning oder das Schwerelosigkeits-Easter-Egg, auf die Sie sonst kaum stoßen würden. Vor allem aber erfahren Sie, warum sich einige Datentypen und Operatoren unerwartet verhalten, und erlangen dadurch ein tieferes Verständnis der Funktionsweise von Python.
- **Kapitel 10: Zweckmäßige Funktionen** Zeigt, wie Sie Funktionen strukturieren sollten, um sie zweckmäßig und übersichtlich zu gestalten. Sie lernen hier die Verwendung von * und ** in der Syntax für Argumente, die Vor- und Nachteile von umfangreichen und kleinen Funktionen sowie funktionale Programmier Techniken wie Lambda-Funktionen kennen.
- **Kapitel 11: Kommentare, Docstrings und Typhinweise** Erklärt, warum die nicht funktionalen Bestandteile von Programmen wichtig sind und wie sie dazu beitragen, dass sich der Code besser pflegen lässt. Sie erfahren hier, wie dicht Kommentare und Docstrings gesetzt sein sollten und wie Sie sie möglichst informativ gestalten können. Außerdem geht es in diesem Kapitel um Typhinweise und um die Verwendung von statischen Analysatoren wie Mypy zum Aufspüren von Bugs.

- **Kapitel 12: Versionssteuerung mit Git** Erklärt, wie Sie mit dem Versionssteuerungssystem Git den Verlauf von Änderungen am Quellcode aufzeichnen und zu früheren Versionen zurückkehren oder das erste Auftreten eines Bugs aufspüren können. Des Weiteren wird beschrieben, wie Cookiecutter Ihnen hilft, die Dateien für Ihre Codeprojekte zu strukturieren.
- **Kapitel 13: Leistungsmessung und Algorithmanalyse** Erklärt, wie Sie die Geschwindigkeit Ihres Codes mithilfe der Module `timeit` und `cProfile` objektiv messen können. Darüber hinaus lernen Sie die Algorithmanalyse mit der O-Notation kennen, mit der Sie vorhersagen können, wie sich die Codeausführung mit zunehmender Datenmenge verlangsamt.
- **Kapitel 14: Praxisprojekte** Hier wenden Sie die in Teil II erlernten Techniken in der Praxis an, indem Sie zwei Befehlszeilenspiele schreiben, nämlich *Turm von Hanoi*, bei dem es darum geht, Scheiben von einem Turm auf einen anderen umzustapeln, und *Vier gewinnt*.

Teil III: Objektorientiertes Python

- **Kapitel 15: Klassen** Erläutert die Bedeutung der objektorientierten Programmierung (OOP), da sie oft missverstanden wird. Viele Entwickler wenden OOP-Techniken im Übermaß an, da sie glauben, dies wäre die übliche Vorgehensweise, und machen ihren Code dadurch unnötig kompliziert. In diesem Kapitel erfahren Sie, wie Klassen geschrieben werden, aber vor allem, wann Sie es tun sollten und wann nicht.
- **Kapitel 16: Vererbung** Erklärt die Vererbung von Klassen und deren praktischen Nutzen für die Wiederverwendung von Code.
- **Kapitel 17: Pythonische OOP: Eigenschaften und Dunder-Methoden** Beschreibt Python-spezifische Merkmale für objektorientierte Programmierung wie Eigenschaften, Dunder-Methoden und Operatorüberladung.

Ihr Weg zur Programmierung

Auf dem Weg vom Anfänger zum erfahrenen Programmierer fühlt man sich oft so, als ob man versucht, aus einem unter hohem Druck stehenden Feuerwehrschauch zu trinken. Angesichts des großen Angebots an Lernstoff sorgen sich viele, ihre Zeit mit ungeeigneten Programmieranleitungen zu vergeuden.

Nachdem Sie dieses Buch gelesen haben (oder vielleicht sogar schon während der Lektüre), sollten Sie sich die folgenden weiteren einführenden Werke ansehen:

- *Python 3 Crashkurs* (dpunkt.verlag, 2020) von Eric Matthes richtet sich zwar an Anfänger, vermittelt dank seines projektgestützten Lehransatzes aber auch erfahrenen Programmierern eine Vorstellung der Python-Bibliotheken Pygame, Matplotlib und Django.
- *Impractical Python Projects* (No Starch Press, 2018) von Lee Vaughan erweitert Ihre Python-Fertigkeiten anhand von Projekten. Die Programme, die Sie aufgrund der Anleitungen in diesem Buch erstellen, machen Spaß und stellen eine großartige Übung dar.
- *Serious Python* (No Starch Press, 2018) von Julien Danjos erklärt, was Sie tun müssen, um sich von einem Hobbyprogrammierer zu einem erfahrenen Softwareentwickler zu mausern, die empfohlenen Vorgehensweisen der Branche zu befolgen und skalierbaren Code zu schreiben.

Die technischen Aspekte sind aber nur eine Stärke von Python. Rund um diese Programmiersprache ist auch eine bunte Community gewachsen. Sie ist für die gut zugängliche Dokumentation und den Support verantwortlich, die unter Programmiersprachen ihres Gleichen suchen. Es gibt jährliche PyCon-Konferenzen sowie viele regionale Zusammenkünfte mit vielfältigen Vorträgen, die sich an Programmierer mit unterschiedlicher Erfahrung richten. Diese Vorträge können Sie sich auch kostenlos online auf <https://pyvideo.org/> ansehen. Um Vorträge zu Ihren Interessengebieten zu finden, schlagen Sie auf der Seite *Tags* nach.

Um sich intensiver mit den anspruchsvolleren Aspekten der Syntax von Python und der Standardbibliothek zu beschäftigen, empfehle ich Ihnen die Lektüre der folgenden Titel:

- *Effektiv Python programmieren* (mitp, 2020) von Brett Slatkin bietet eine beeindruckende Zusammenstellung von empfohlenen *pythonischen* Vorgehensweisen und Sprachmerkmalen.
- *Python Cookbook* (O'Reilly Media, 2013) von David Beazley und Brian K. Jones wartet mit einer umfangreichen Menge von Codebausteinen auf, mit denen Python-Neulinge ihr Repertoire erweitern können.
- *Fluent Python* (O'Reilly Media, 2021) von Luciano Ramalho ist ein Meisterwerk, das die Feinheiten von Python erklärt. Sein Umfang von fast 800 Seiten mag abschreckend wirken, aber die Lektüre ist mit Sicherheit der Mühe wert.

Viel Glück auf Ihrem Weg zur besseren Programmierung! Fangen wir an!

1

Fehlermeldungen und Recherche



Vermenschlichen Sie niemals Computer – das können die nämlich überhaupt nicht leiden! Aber im Ernst: Wenn Ihnen ein Computer eine Fehlermeldung präsentiert, dann liegt das nicht daran, dass Sie ihn beleidigt hätten. Computer sind zwar die anspruchsvollsten Werkzeuge, mit denen die meisten von uns jemals zu tun bekommen, aber sie sind und bleiben nun mal nichts anderes als Werkzeuge.

Dennoch sind wir schnell geneigt, diesen Werkzeugen Schuld zuzuschieben. Wenn Sie programmieren lernen, sind Sie dabei zum größten Teil auf sich selbst gestellt. Dabei kann man sich schnell als Versager fühlen, wenn man immer noch mehrmals täglich im Internet nachforscht, auch wenn man sich bereits seit Monaten mit Python beschäftigt. Aber selbst professionelle Softwareentwickler suchen im Internet oder schlagen in der Dokumentation nach, um Fragen zur Programmierung zu klären.

Sofern Sie nicht über die Geldmittel oder die guten Kontakte verfügen, um einen Privatlehrer zu engagieren, der Ihnen alle Fragen rund ums Programmieren beantwortet, stehen Ihnen nur Ihr Computer, die Suchmaschinen im Internet und

Ihre eigene Geschicklichkeit zur Verfügung. Glücklicherweise aber wurden die Fragen, die Sie haben, mit hoher Wahrscheinlichkeit schon einmal gestellt. Für Programmierer ist es viel wichtiger, selbstständig Antworten zu finden, als irgendwelche Algorithmen oder Datenstrukturen zu kennen. In diesem Kapitel erfahren Sie, wie Sie sich diese unverzichtbare Fähigkeit aneignen können.

Python-Fehlermeldungen verstehen

Bei der Konfrontation mit einer Fehlermeldung, die einen Riesenschwall von Technobla enthält, neigen viele Programmierer in einem ersten Impuls heraus dazu, sie völlig zu ignorieren. Allerdings versteckt sich in dieser Meldung die Antwort auf die Frage, was bei dem Programm schief läuft. Um diese Antwort zu finden, sind zwei Schritte erforderlich: Sie müssen die Ablaufverfolgung (Traceback) untersuchen und die Fehlermeldung im Internet recherchieren.

Tracebacks untersuchen

Python-Programme stürzen ab, wenn der Code eine Ausnahme auslöst, die nicht von einer `except`-Anweisung behandelt wird. Wenn das geschieht, zeigt Python die Meldung dieser Ausnahme und ein *Traceback* (oder auch *Stacktrace* oder *Ablaufverfolgung*) an. Darin ist die Stelle in dem Programm angegeben, an der die Ausnahme ausgelöst wurde, sowie der Ablauf der Funktionsaufrufe, die dorthin geführt haben.

Um das Lesen von Tracebacks zu üben, geben Sie das folgende fehlerhafte Programm ein und speichern es als `abcTraceback.py`. Die Zeilennummern dienen hier nur zur Orientierung und gehören nicht mit zu dem Programm.

```
1. def a():
2.     print('Start of a()')
3.     b() # Ruft b() auf.      ❶
4.
5. def b():
6.     print('Start of b()')
7.     c() # Ruft c() auf.      ❷
8.
9.     def c():
10.        print('Start of c()')
11.        42 / 0 # Verursacht eine Division durch null.    ❸
12.
13. a() # Ruft a() auf.
```

In diesem Programm ruft `a()` die Funktion `b()` auf ❶ und diese wiederum `c()` ❷. Innerhalb von `c()` aber führt der Ausdruck `42 / 0` ❸ zu einem Fehler aufgrund der Division durch null. Wenn Sie dieses Programm ausführen, erhalten Sie folgende Ausgabe:

```
Start of b()
Start of c()
Traceback (most recent call last):
  File "abcTraceback.py", line 13, in <module>
    a() # Ruft a() auf.
  File "abcTraceback.py", line 3, in a
    b() # Ruft b() auf.
  File "abcTraceback.py", line 7, in b
    c() # Ruft c() auf.
  File "abcTraceback.py", line 11, in c
    42 / 0 # Verursacht eine Division durch null.
ZeroDivisionError: division by zero
```

Sehen wir uns diese Ablaufverfolgung nun Zeile für Zeile an. Dabei fangen wir mit der folgenden Zeile an:

```
Traceback (most recent call last):
```

Diese Meldung besagt, dass als Nächstes ein `Traceback` folgt. Die Angabe `most recent call last` bedeutet, dass die Funktionsaufrufe in chronologischer Reihenfolge aufgeführt werden, also von der zuerst bis zu der zuletzt aufgerufenen Funktion.

Die nächste Zeile gibt dann den ersten Funktionsaufruf aus:

```
File "abcTraceback.py", line 13, in <module>
    a() # Ruft a() auf.
```

Diese beiden Zeilen bilden eine *Frame-Übersicht* und geben die Informationen an, die sich in einem *Frame-Objekt* befinden. Beim Aufruf einer Funktion werden die Daten der lokalen Variablen sowie die Stelle im Code, zu der nach dem Ende des Funktionsaufrufs zurückgesprungen werden soll, in einem solchen Objekt gespeichert. *Frame-Objekte* enthalten also die Daten der lokalen Variablen und weitere Daten, die im Zusammenhang mit einem Funktionsaufruf stehen. Sie werden erstellt, wenn eine Funktion aufgerufen wird, und zerstört, wenn die Funktion die Steuerung zurückgibt. Das `Traceback` enthält eine *Frame-Übersicht* für jeden *Frame* bis zum Absturz. Dieser Übersicht können wir entnehmen, dass der Funktionsaufruf in Zeile 13 von `abcTraceback.py` erfolgte. Die Angabe `<module>` teilt uns außerdem mit, dass sich die Zeile im globalen Gültigkeitsbereich befindet. Mit einer Einrückung um zwei Stellen wird außerdem die Zeile 13 ausgegeben.

Die nächsten vier Zeilen enthalten zwei weitere Frame-Übersichten:

```
File "abcTraceback.py", line 3, in a
    b() # Ruft b() auf.
File "abcTraceback.py", line 7, in b
    c() # Ruft c() auf.
```

Die Angabe `line 3, in a` sagt uns, dass `b()` in Zeile 3 innerhalb der Funktion `a()` aufgerufen wurde. Das wiederum führte dazu, dass in Zeile 7 innerhalb von `b()` die Funktion `c()` aufgerufen wurde. Beachten Sie, dass die Aufrufe von `print()` in Zeile 2, 6 und 10 im Traceback nicht erscheinen, obwohl sie vor den Funktionsaufrufen ausgeführt wurden. In einer Ablaufverfolgung werden nur die Zeilen mit den Funktionsaufrufen bis zum Auslösen der Ausnahme ausgegeben.

Die letzte Frame-Übersicht gibt die Zeile an, in der die nicht behandelte Ausnahme ausgelöst wurde, gefolgt vom Namen und der Meldung dieser Ausnahme:

```
File "abcTraceback.py", line 11, in c
    42 / 0 # Verursacht eine Division durch null.
ZeroDivisionError: division by zero
```

Die angegebene Nummer ist diejenige der Zeile, in der Python den Fehler bemerkt hat. Die eigentliche Ursache des Fehlers kann sich jedoch durchaus weiter vorn befinden.

Fehlermeldungen sind berüchtigt für ihre Kürze und Rätselhaftigkeit. Die drei Wörter `division by zero` werden Ihnen nicht viel sagen, sofern Sie nicht wissen, dass eine Division durch null mathematisch unmöglich ist und einen häufigen Bug in Software darstellt. In diesem Programm ist der Fehler nicht sehr schwer zu finden. Ein Blick auf die Codezeile in der Frame-Übersicht genügt, um zu erkennen, wo in dem Code `42 / 0` die Division durch null erfolgt.

Sehen wir uns nun aber ein kniffligeres Beispiel an. Geben Sie den folgenden Code in einen Texteditor ein und speichern Sie ihn als `zeroDivideTraceback.py`:

```
def spam(number1, number2):
    return number1 / (number2 - 42)

spam(101, 42)
```

Wenn Sie dieses Programm ausführen, erhalten Sie folgende Ausgabe:

```
Traceback (most recent call last):
  File "zeroDivideTraceback.py", line 4, in <module>
    spam(101, 42)
  File "zeroDivideTraceback.py", line 2, in spam
    return number1 / (number2 - 42)
ZeroDivisionError: division by zero
```

Die Fehlermeldung ist die gleiche wie zuvor, aber die Division durch null in `return number1 / (number2 - 42)` lässt sich nicht so leicht erkennen. Der Operator `/` zeigt Ihnen, dass hier eine Division erfolgt, und offensichtlich wird der Ausdruck `(number2 - 42)` zu 0 ausgewertet. Daraus können Sie schließen, dass die Funktion `spam()` fehlschlägt, wenn der Parameter `number2` auf 42 gesetzt wird.

Manchmal gibt ein `Traceback` auch eine Zeile aus, die hinter der eigentlichen Ursache des Bugs liegt. Betrachten Sie dazu das folgende Programm, bei dem in der ersten Zeile die schließende Klammer fehlt:

```
print('Hello.'  
print('How are you?')
```

Die Fehlermeldung aber weist auf ein Problem in der zweiten Zeile hin:

```
File "example.py", line 2  
    print('How are you?')  
    ^  
SyntaxError: invalid syntax
```

Der Grund dafür ist, dass der Python-Interpreter den Syntaxfehler erst bemerkt, wenn er die zweite Zeile liest. Das `Traceback` zeigt Ihnen, wo etwas schiefgeht, aber dort liegt nicht immer die Ursache des Fehlers. Wenn die `Frame`-Übersicht Ihnen nicht genügend Informationen gibt, um den Bug zu erkennen, oder wenn die wahre Ursache in einer vorherigen Zeile steckt, dann müssen Sie das Programm mit einem Debugger durchgehen oder Protokollmeldungen durchsuchen, um den Fehler zu finden. Eine Internetrecherche der Fehlermeldung kann Ihnen dabei entscheidende Hinweise geben.

Fehlermeldungen recherchieren

Fehlermeldungen sind sehr kurz und oft nicht einmal ganze Sätze. Da Programmierer sie häufiger zu Gesicht bekommen, sind sie eher als Gedächtnisstützen gedacht und weniger als ausführliche Erklärungen. Wenn Sie eine neue Fehlermeldung zum ersten Mal sehen, können Sie sie kopieren und in eine Suchmaschine einfügen. Dadurch erhalten Sie oft ausführliche Erklärungen darüber, was die Meldung besagt und was die möglichen Ursachen sein können. Abbildung 1–1 zeigt die Ergebnisse einer Suche nach *python »ZeroDivisonError: division by zero«*. Die Anführungszeichen sorgen dafür, dass Vorkommen der genauen Formulierung gefunden werden, und der Zusatz *python* engt die Suche sinnvoll ein.

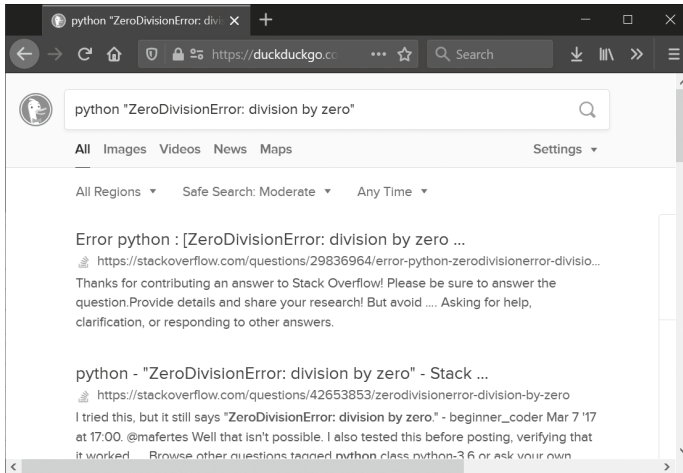


Abb. 1–1 Durch das Kopieren einer Fehlermeldung in eine Suchmaschine können Sie Erklärungen und Lösungsvorschläge schnell finden.

Fehlermeldungen zu recherchieren, hat nichts mit Schummeln zu tun. Niemand kann sich sämtliche Fehlermeldungen merken, die in einer Programmiersprache auftreten können. Professionelle Softwareentwickler nutzen täglich das Internet, um Fragen zur Programmierung zu klären.

Es ist sinnvoll, bei der Suche den Teil der Fehlermeldung auszuschließen, der sich ausschließlich auf Ihren Code bezieht. Betrachten Sie zum Beispiel die folgenden Fehlermeldungen:

```
>>> print(employeRecord)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'employeRecord' is not defined      ❶
>>> 42 - 'hello'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'int' and 'str'      ❷
```

In diesem Beispiel wurde eine Variable als `employeRecord` falsch geschrieben, was einen Fehler hervorruft ❶. Da der Bezeichner `employeRecord` in der Meldung `NameError: name 'employeRecord' is not defined` nur in Ihrem Code eine Bedeutung hat, sollten Sie nicht nach dem gesamten Text der Meldung suchen, sondern nur nach `python »NameError: name« »is not defined«`. In der letzten Zeile beziehen sich die Angaben `'int'` und `'str'` in der Fehlermeldung ❷ auf die Werte `42` und `'hello'`, weshalb es sinnvoll ist, nur nach `python »TypeError: unsupported operand type(s)«` zu suchen. Wenn Sie mit solchen verkürzten Suchbegriffen keinen Erfolg haben, können Sie versuchen, nach dem kompletten Text der Fehlermeldung zu forschen.