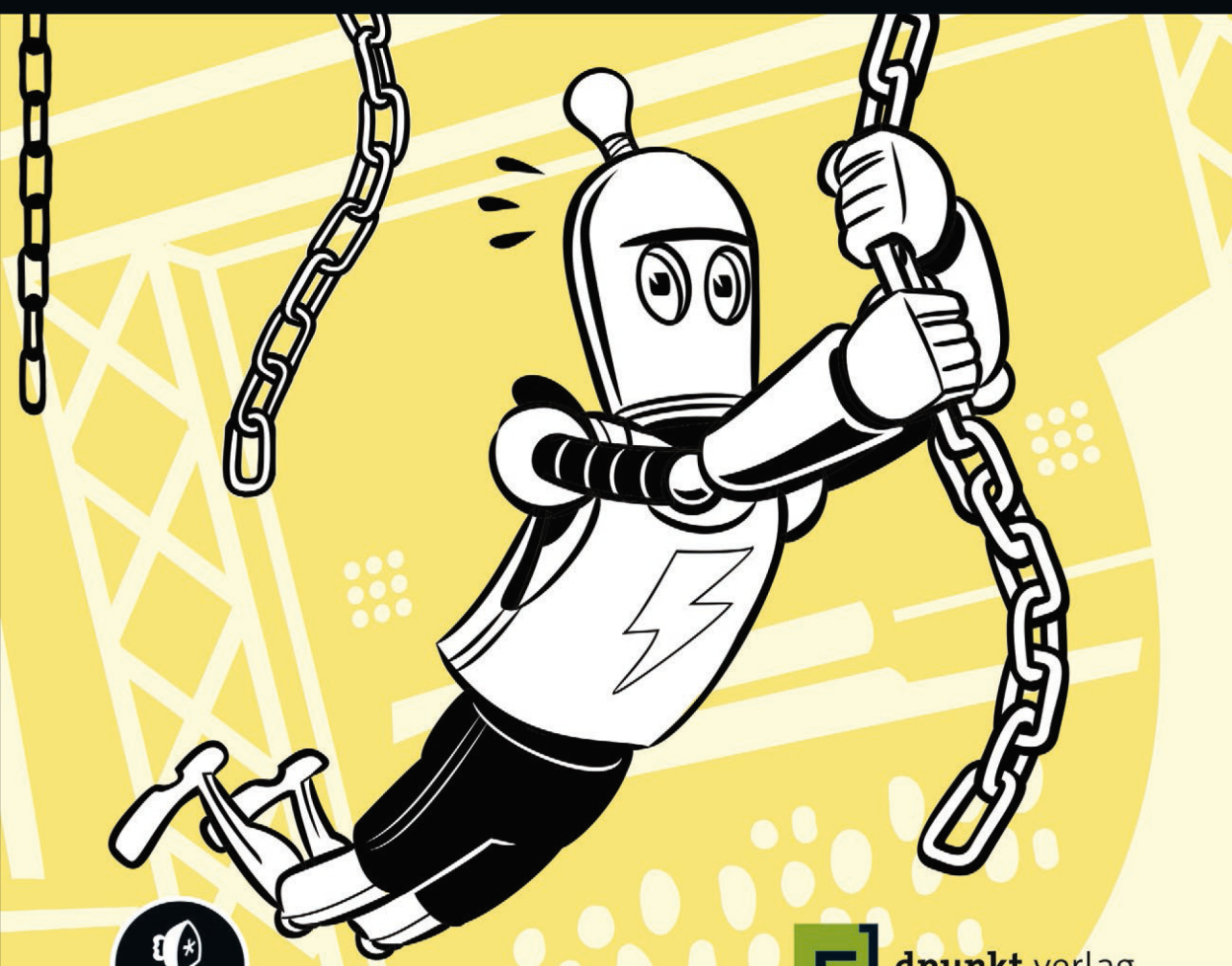
Der Weg zum Python-Profi

Ein Best-Practice-Buch für sauberes Programmieren

Al Sweigart



no starch press

dpunkt.verlag

Al Sweigart

Der Weg zum Python-Profi

Ein Best-Practice-Buch für sauberes Programmieren

dpunkt.verlag

Al Sweigart

Lektorat: Gabriel Neumann
Lektoratsassistenz: 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

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.

PEFC zertifiziert
Das Papier für dieses Buch stammt aus nachhaltig bewirtschafteten Wäldern und kontrollierten Quellen.
PEFC™
PEFC/04-31-0810
www.pefc.de

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

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 *Coole 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 mitzuhalten. 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 durchzuarbeiten, 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 Programmiertechniken 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 gesät 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 Algorithmusanalyse** Erklärt, wie Sie die Geschwindigkeit Ihres Codes mithilfe der Module `timeit` und `cProfile` objektiv messen können. Darüber hinaus lernen Sie die Algorithmusanalyse 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 Feuerwehrschlauch 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.

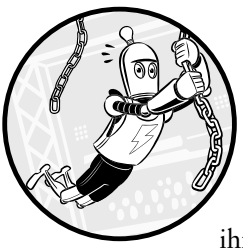
Um sich intensiver mit den anspruchsvolleren Aspekten der Syntax von Python und der Standardbibliothek zu beschäftigen, empfehle ich Ihnen die Lektüre der folgenden Titel:

- *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 Riesenwust von Technoblabla 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 schiefläuft. 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 *abcTracebyck.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 *zeroDevideTraceback.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.

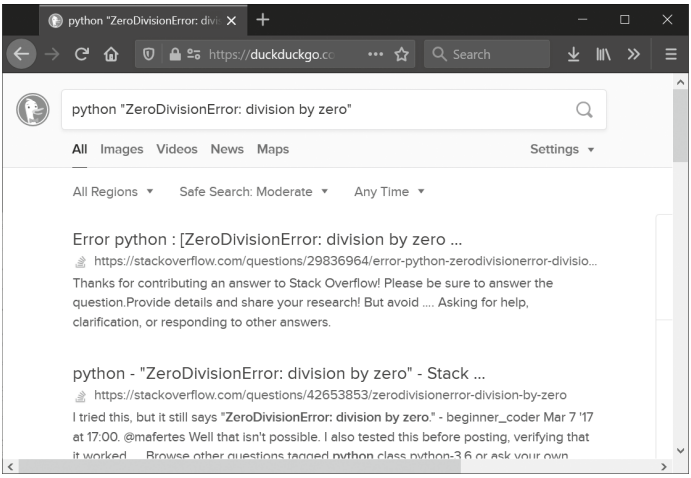


Abb. 1–1 *Durch das Kopieren einer Fehlermeldung in eine Suchmaschine können Sie Erklärungen und Lösungsvorschläge schnell finden.*

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.