

3. AUFLAGE

PYTHON CRASHKURS

EINE PRAKTISCHE, PROJEKTBASIERTE
PROGRAMMIEREINFÜHRUNG

ERIC MATTHES

WELTWEIT
1,5 MIO. VERKAUFTE
EXEMPLARE



dpunkt.verlag

Vorwort zur dritten Auflage

Die Reaktion auf die zweite Auflage von *Python Crashkurs* war überwältigend positiv. Es wurden einschließlich der Übersetzungen in über 10 Sprachen mehr als eine Million Exemplare gedruckt. Ich erhielt Briefe und E-Mails von Lesern, sowohl von Zehnjährigen als auch von Rentnern, die in ihrer Freizeit programmieren lernen möchten. *Python Crashkurs* wird im Sekundarstufenunterricht, aber auch in Hochschulkursen eingesetzt. Studenten, an die anspruchsvollere Lehrbücher ausgeteilt wurden, haben *Python Crashkurs* als ergänzenden Text für ihren Kurs benutzt und waren sehr zufrieden damit. Das Buch wird verwendet, um die Programmierfähigkeit für den Beruf zu verbessern, umzusteigen und um an Privatprojekten zu arbeiten. Kurzum, es wird so vielfältig genutzt, wie ich es mir gewünscht habe.

Es war überaus erfreulich, eine dritte Auflage von *Python Crashkurs* zu schreiben. Auch wenn Python eine ausgereifte Sprache ist, entwickelt sie sich wie jede andere Sprache weiter. Mein Hauptziel bei der Überarbeitung des Buches war es, einen gut zusammengestellten Python-Einführungskurs zu bieten. Durch die Lektüre dieses Buches lernen Sie alles, was Sie brauchen, um mit der Arbeit an Ihren eigenen Projekten zu beginnen, und Sie werden eine solide Grundlage für all Ihre zukünftigen Fortschritte schaffen. Ich habe einige Abschnitte auf den neuesten

Stand gebracht, um neue, einfachere Möglichkeiten darzustellen, die es jetzt zur Erledigung von Aufgaben in Python gibt. Außerdem habe ich einige Abschnitte deutlicher formuliert, in denen Aspekte der Sprache nicht klar genug dargestellt wurden. Alle Projekte wurden unter Zuhilfenahme weitverbreiteter, gut gepflegter Bibliotheken komplett überarbeitet, die Sie auch vertrauensvoll für Ihre eigenen Projekte einsetzen können.

Der folgende Überblick führt besondere Änderungen auf, die ich in dieser dritten Auflage vorgenommen habe:

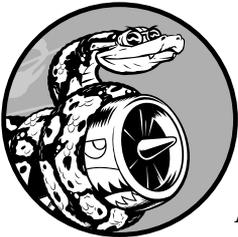
- Kapitel 1 enthält jetzt den Texteditor VS Code, der bei Anfängern und professionellen Programmierern beliebt ist und auf allen Betriebssystemen problemlos funktioniert.
- Kapitel 2 enthält die neuen Methoden `removeprefix()` und `removesuffix()`, die bei der Arbeit mit Dateien und URLs hilfreich sind. In diesem Kapitel finden Sie auch die neuen, verbesserten Fehlermeldungen von Python, die Ihnen bei der Fehlersuche in Ihrem Code helfen, wenn etwas schief läuft.
- Kapitel 10 verwendet das Modul `pathlib` für die Arbeit mit Dateien. Dies ist ein viel einfacherer Ansatz zum Lesen aus und Schreiben in Dateien.
- Kapitel 11 verwendet `pytest`, um automatisierte Tests für den von Ihnen geschriebenen Code zu schreiben. Die `pytest`-Bibliothek hat sich zum Standardwerkzeug für das Schreiben von Tests in Python entwickelt. Sie ist so benutzerfreundlich, dass Sie sie für Ihre ersten Tests verwenden können; und wenn Sie eine Karriere als Python-Programmierer anstreben, werden Sie sie auch im professionellen Umfeld einsetzen.
- Das Projekt *Alien Invasion* in den Kapiteln 12–14 enthält eine Einstellung zur Steuerung der Bildwiederholrate, die dafür sorgt, dass das Spiel auf verschiedenen Betriebssystemen gleichmäßig läuft. Für den Aufbau der Alien-Flotte wird ein einfacherer Ansatz verwendet, und auch die Gesamtorganisation des Projekts wurde aufgeräumt.
- Die Datenvisualisierungsprojekte in den Kapiteln 15–17 verwenden die neuesten Funktionen von `Matplotlib` und `Plotly`. Die `Matplotlib`-Visualisierungen verfügen über aktualisierte Style-Einstellungen. Das Random-Walk-Projekt weist eine kleine Verbesserung auf, die die Genauigkeit der Diagramme erhöht, was bedeutet, dass Sie bei jedem neuen Durchlauf eine größere Vielfalt an Mustern sehen. Alle Projekte mit `Plotly` verwenden jetzt das Modul `Plotly Express`, mit dem Sie Ihre ersten Visualisierungen mit nur wenigen Zeilen Code erstellen können. Sie können ganz einfach eine Vielzahl von Visualisierungen ausprobieren, bevor Sie sich auf eine Art von Plot festlegen, und sich dann auf die Verfeinerung einzelner Elemente dieses Plots konzentrieren.

- Das Projekt Learning Log in den Kapiteln 18–20 wurde mit der neuesten Version von Django erstellt und mit der neuesten Version von Bootstrap gestylt. Einige Teile des Projekts wurden umbenannt, um die Gesamtorganisation des Projekts leichter nachvollziehen zu können. Das Projekt wird jetzt auf Platform.sh bereitgestellt, einem modernen Hosting-Service für Django-Projekte. Der Bereitstellungsprozess wird durch YAML-Konfigurationsdateien gesteuert, die Ihnen eine große Gestaltungsfreiheit geben, wie Ihr Projekt bereitgestellt wird. Dieser Ansatz steht im Einklang mit der Art und Weise, wie professionelle Programmierer moderne Django-Projekte bereitstellen.
- Anhang A wurde vollständig aktualisiert, um aktuelle Methoden für die Installation von Python auf allen wichtigen Betriebssystemen zu empfehlen. Anhang B enthält eine detaillierte Anleitung zum Einrichten von VS Code und kurze Beschreibungen der wichtigsten Texteditoren und IDEs, die derzeit verwendet werden. Anhang C verweist die Leser auf einige der beliebtesten Onlineressourcen, um Hilfe zu erhalten. Anhang D bietet weiterhin einen Mini-Crashkurs in der Verwendung von Git für die Versionskontrolle. Anhang E ist ganz neu in der dritten Auflage. Selbst mit einer guten Anleitung für die Bereitstellung der von Ihnen erstellten Anwendungen können viele Dinge schiefgehen. Dieser Anhang bietet eine detaillierte Anleitung zur Fehlerbehebung, die Sie verwenden können, wenn der Bereitstellungsprozess nicht auf Anhieb funktioniert. Der Index wurde gründlich überarbeitet und ermöglicht es Ihnen, dieses Buch auch als Nachschlagewerk für Ihre zukünftigen Python-Projekte zu verwenden.

Vielen Dank dafür, dass Sie *Python Crashkurs* lesen! Wenn Sie irgendwelche Rückmeldungen oder Fragen an mich haben, können Sie sich gern an mich wenden, auf Twitter bin ich *@ehmatthes*.

2

Variablen und einfache Datentypen



In diesem Kapitel lernen Sie die verschiedenen Arten von Daten kennen, mit denen Sie in Python-Programmen arbeiten können. Außerdem erfahren Sie, wie Sie die Daten in Variablen speichern und diese Variablen in Ihren Programmen einsetzen.

Was bei der Ausführung von `hello_world.py` wirklich geschieht

Sehen wir uns nun genauer an, was Python macht, wenn Sie `hello_world.py` laufen lassen. Selbst bei einem so einfachen Programm erledigt Python eine ziemliche Menge an Arbeit:

```
print("Hello Python world!")
```

```
hello_world.py
```

Wenn Sie diesen Code ausführen, erhalten Sie folgende Ausgabe:

```
Hello Python world!
```

Die Dateiendung `.py` gibt an, dass es sich bei der Datei um ein Python-Programm handelt. Der Editor führt die Datei mithilfe des *Python-Interpreters* aus, der das Programm liest und dabei ermittelt, was jedes einzelne darin enthaltene Wort bedeutet. Sieht er beispielsweise das Wort `print`, gefolgt von Klammern, gibt er den Inhalt dieser Klammern auf dem Bildschirm aus.

Während Sie ein Programm schreiben, kennzeichnet der Editor die verschiedenen Teile des Programms durch unterschiedliche Farben. Beispielsweise erkennt er, dass es sich bei `print()` um den Namen einer Funktion handelt, und stellt das Wort in einer bestimmten Farbe dar. Er bemerkt auch, dass "Hello Python world!" kein Python-Code ist, und färbt diesen Satz daher anders ein. Diese sogenannte *Syntaxkennzeichnung* ist sehr praktisch, wenn Sie Ihre eigenen Programme schreiben.

Variablen

Versuchen wir nun, in `hello_world.py` eine Variable zu verwenden. Fügen Sie am Anfang der Datei eine neue Zeile ein und ändern Sie die zweite Zeile:

```
message = "Hello Python world!"  
print(message)
```

`hello_world.py`

Wenn Sie dieses Programm ausführen, erhalten Sie die gleiche Ausgabe wie zuvor:

```
Hello Python world!
```

Was wir hinzugefügt haben, ist eine *Variable* namens `message`. Jede Variable enthält einen *Wert*, also eine Information, die mit ihr verknüpft ist. In diesem Fall ist der Wert der Text "Hello Python world!".

Die Ergänzung um eine Variable macht dem Python-Interpreter etwas mehr Arbeit. Wenn er die erste Zeile verarbeitet, verknüpft er den Text "Hello Python world!" mit der Variablen `message`. In der zweiten Zeile angekommen, gibt er den Wert von `message` auf dem Bildschirm aus.

Wir wollen das Programm nun noch um eine zweite Meldung erweitern. Fügen Sie am Ende von `hello_world.py` eine leere Zeile und dann zwei neue Codezeilen ein:

```
message = "Hello Python world!"  
print(message)
```

```
message = "Hello Python Crash Course world!"  
print(message)
```

Wenn Sie jetzt *hello_world.py* ausführen, erhalten Sie eine Ausgabe von zwei Zeilen:

```
Hello Python world!  
Hello Python Crash Course world!
```

Den Wert einer Variablen können Sie in einem Programm jederzeit ändern. Python merkt sich stets den aktuellen Wert.

Variablen benennen und verwenden

Wenn Sie in Python Variablen verwenden, müssen Sie einige Regeln und Richtlinien einhalten. Die Richtlinien dienen nur dazu, Code zu schreiben, der sich leichter lesen und verstehen lässt, doch wenn Sie die Regeln brechen, kann das einen Fehler zur Folge haben. Merken Sie sich daher folgende Punkte für den Umgang mit Variablen:

- Variablennamen dürfen nur Buchstaben, Ziffern und Unterstriche enthalten. Am Anfang darf ein Buchstabe oder ein Unterstrich stehen, aber keine Zahl. Beispielsweise können Sie eine Variable `message_1` nennen, aber nicht `1_message`.
- Leerzeichen sind in Variablennamen nicht erlaubt. Um Namensbestandteile zu trennen, können Sie Unterstriche verwenden. Beispielsweise ist `greeting_message` ein gültiger Variablenname, wohingegen `greeting message` einen Fehler hervorruft.
- Verwenden Sie keine Python-Schlüsselwörter und -Funktionsnamen als Variablennamen, also keine Wörter, die in Python für bestimmte Zwecke reserviert sind, z. B. das Wort `print` (siehe »Schlüsselwörter und integrierte Funktionen« in Anhang A).
- Variablennamen sollten kurz, aber aussagekräftig sein. Beispielsweise ist `name` besser als `n`, `student_name` besser als `s_n` und `name_length` besser als `length_of_persons_name`.
- Seien Sie bei der Verwendung des kleinen `l` und des großen `O` vorsichtig, da diese Buchstaben leicht mit den Zahlen `1` und `0` verwechselt werden können.

Es kann etwas Übung erfordern, sich gute Variablennamen auszudenken, vor allem, wenn Ihre Programme interessanter und komplizierter werden. Je mehr Programme Sie schreiben und je mehr Code anderer Leute Sie lesen, umso besser werden Sie jedoch darin, Variablen aussagekräftig zu benennen.



Hinweis

Momentan noch sollten Sie in Python ausschließlich Variablennamen in Kleinbuchstaben verwenden. Großbuchstaben rufen zwar keine Fehler hervor, haben aber eine besondere Bedeutung, die Sie in späteren Kapiteln noch kennenlernen werden.

Fehler bei Variablennamen vermeiden

Alle Programmierer machen Fehler, die meisten sogar täglich. Gute Programmierer wissen allerdings, wie sie diese Fehler ohne großen Aufwand beheben können. Als Beispiel wollen wir uns einen Fehler ansehen, wie er in der Anfangsphase häufig vorkommt, und zeigen, wie Sie ihn korrigieren können.

Zur Veranschaulichung schreiben wir Code, der absichtlich einen Fehler hervorruft. Geben Sie den folgenden Code ein – einschließlich des fett hervorgehobenen, falsch geschriebenen Wortes `message`:

```
message = "Hello Python Crash Course reader!"
print(message)
```

Wenn in einem Programm ein Fehler auftritt, versucht der Python-Interpreter sein Bestes, um Ihnen dabei zu helfen, die problematische Stelle zu finden. Dazu gibt er eine *Rückverfolgung* oder ein *Traceback* aus, wenn sich ein Programm nicht erfolgreich ausführen lässt. Dabei handelt es sich um eine Angabe der Stelle, an der der Interpreter auf Schwierigkeiten gestoßen ist. Bei unserem Beispielcode mit dem falsch geschriebenen Variablennamen gibt Python folgendes Traceback aus:

```
Traceback (most recent call last):
❶ File "hello_world.py", line 2, in <module>
❷   print(message)
   ~~~~~
❸ NameError: name 'message' is not defined. Did you mean: 'message'?
```

Die Ausgabe bei ❶ zeigt, dass der Fehler in Zeile 2 der Datei `hello_world.py` aufgetreten ist. Außerdem zeigt der Interpreter die betreffende Zeile an, sodass wir den Fehler schneller erkennen können (❷), und er gibt an, um was für eine Art Fehler es sich handelt (❸). In diesem Fall ist ein *Namensfehler aufgetreten*, denn die auszugebende Variable `message` ist nicht definiert. Das heißt, Python kann den angegebenen Variablennamen nicht erkennen. Ein Namensfehler bedeutet entweder, dass wir vergessen haben, einen Wert für die Variable festzulegen, bevor wir sie verwenden, oder dass wir den Namen der Variablen falsch geschrieben haben. Wenn Python einen Variablennamen findet, der dem nicht erkannten ähnlich ist, fragt es, ob das der Name ist, den Sie verwenden wollten.

In diesem Fall ist der Fehler offensichtlich: Wir haben in der zweiten Zeile ein `s` im Variablennamen `message` vergessen. Der Python-Interpreter führt an unserem Code keine Rechtschreibprüfung durch, verlangt aber, dass die Variablennamen einheitlich geschrieben sind. Schauen Sie sich an, was passiert, wenn wir `message` beide Male falsch schreiben:

```
mesage = "Hello Python Crash Course reader!"  
print(mesage)
```

In diesem Fall wird das Programm erfolgreich ausgeführt:

```
Hello Python Crash Course reader!
```

Da die Namen der Variablen übereinstimmen, sieht Python keine Probleme. Programmiersprachen sind streng, können aber nicht zwischen guter und schlechter Rechtschreibung unterscheiden. Daher kommt es bei Variablennamen und Code nicht auf die englischen oder deutschen Rechtschreib- und Grammatikregeln an.

Bei vielen Programmierfehlern handelt es sich um einfache Tippfehler in einer einzigen Programmzeile. Wenn Sie viel Zeit damit verbringen, solche Fehler aufzuspüren, lassen Sie sich gesagt sein, dass Sie sich in bester Gesellschaft befinden. Viele erfahrene und talentierte Programmierer sind Stunden damit beschäftigt, solche winzigen Fehler auszumerzen. Versuchen Sie, über solche Missgeschicke zu lachen und weiterzumachen. Sie werden Ihnen im Programmiererleben noch häufig unterlaufen.

Variablen sind Etiketten

Variablen werden oft mit Kisten verglichen, in denen Sie Werte ablegen können. Dieses Bild kann hilfreich sein, wenn Sie beginnen, Variablen zu verwenden, beschreibt aber nicht genau, wie Variablen intern in Python dargestellt werden. Es ist besser, sich Variablen als Etiketten vorzustellen, denen Sie Werte zuweisen. Sie können auch sagen, dass eine Variable auf einen bestimmten Wert verweist.

Bei den ersten Programmen, die Sie schreiben, wird dieser Unterschied wahrscheinlich nicht viel ausmachen, aber je früher Sie sich darüber im Klaren sind, umso besser. Mit Sicherheit werden Sie irgendwann einmal mit dem überraschenden Verhalten einer Variablen konfrontiert. Ein genaues Verständnis der tatsächlichen Funktionsweise von Variablen kann Ihnen dann helfen zu erkennen, was in Ihrem Code vor sich geht.



Hinweis

Um neue Programmierprinzipien zu verstehen, ist es am besten, sie in eigenen Programmen anzuwenden. Wenn Sie bei einer Aufgabe in diesem Buch nicht weiterkommen, versuchen Sie, zunächst einmal etwas anderes zu tun. Wenn Sie immer noch nicht weiterkommen, schauen Sie sich noch einmal die entsprechenden Abschnitte in dem Kapitel an. Wenn Sie Hilfe brauchen, können Sie in Anhang C erfahren, an wen Sie sich wenden können.

Probieren Sie es selbst aus!

Schreiben Sie für jede einzelne Übung ein eigenes Programm und speichern Sie diese Programme jeweils mit einem Dateinamen, der der Python-Konvention genügt, der also nur aus Kleinbuchstaben und Unterstrichen besteht, z. B. *simple_message.py* und *simple_messages.py*.

2-1 Einfache Nachricht: Weisen Sie eine Nachricht einer Variablen zu und geben Sie sie aus.

2-2 Einfache Nachrichten: Weisen Sie eine Nachricht einer Variablen zu und geben Sie sie aus. Ändern Sie dann den Wert der Variablen in eine neue Nachricht und geben Sie diese ebenfalls aus.

Strings

Die meisten Programme definieren und erfassen Daten einer bestimmten Art und machen dann irgendetwas Sinnvolles damit. Daher ist es hilfreich, die verschiedenen Arten von Daten in Kategorien einzuteilen. Der erste Datentyp, den wir uns dabei ansehen, ist der String. Strings erscheinen auf den ersten Blick ganz einfach, lassen sich aber sehr vielseitig verwenden.

Bei einem *String* handelt es sich um eine Folge von Zeichen. In Python wird alles, was in Anführungszeichen eingeschlossen ist, als String aufgefasst. Dabei können Sie sowohl einfache als auch doppelte Anführungszeichen verwenden:

```
"Dies ist ein String."
'Dies ist auch ein String.'
```

Das ermöglicht es auch, innerhalb von Strings Anführungszeichen und Apostrophe zu verwenden:

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

Sehen wir uns noch weitere Möglichkeiten zur Verwendung von Strings an.

Groß- und Kleinschreibung mithilfe von Methoden ändern

Eine der einfachsten Aufgaben bei der Arbeit mit Strings besteht darin, die Groß- und Kleinschreibung zu ändern. Schauen Sie sich den folgenden Code an und versuchen Sie herauszufinden, was passiert:

```
name = "ada lovelace"
print(name.title())
```

name.py

Speichern Sie die Datei als *name.py* und führen Sie sie aus. Dabei erhalten Sie folgende Ausgabe:

```
Ada Lovelace
```

In diesem Beispiel verweist die Variable `name` auf den kleingeschriebenen String `"ada lovelace"`. Im Aufruf von `print()` steht aber hinter der Variablen noch die Methode `title()`. Eine *Methode* ist eine Aktion, die Python an Daten ausführen kann. Der Punkt in `name.title()` weist Python an, die Methode `title()` auf die Variable `name` anzuwenden. Hinter einem Methodennamen steht immer ein Klammernpaar, da viele Methoden zusätzliche Angaben benötigen, um ihre Aufgaben ausführen zu können. Diese Angaben werden dann in die Klammern geschrieben. Die Funktion `title()` braucht jedoch keine zusätzlichen Angaben, weshalb die Klammern hier leer sind.

Die Methode `title()` zeigt alle Wörter mit großem Anfangsbuchstaben an. Das ist praktisch, wenn Ihr Programm etwa die Eingabewerte `Ada`, `ADA` und `ada` alle als denselben Namen erkennen und stets als `Ada` ausgeben soll.

Es gibt noch mehrere andere praktische Methoden, um die Groß- und Kleinschreibung zu ändern. Beispielsweise können Sie einen String wie folgt komplett in Groß- oder Kleinbuchstaben umwandeln:

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

Dies führt zu folgender Ausgabe:

```
ADA LOVELACE
ada lovelace
```

Die Methode `lower()` ist insbesondere für die Speicherung von Daten nützlich. Oft können Sie nicht darauf vertrauen, dass die Benutzer Ihres Programms die Groß- und Kleinschreibung richtig machen, weshalb Sie alle eingegebenen Strings vor der Speicherung in Kleinbuchstaben umwandeln. Wenn Sie die Informationen

anzeigen wollen, können Sie dann die Art von Groß- und Kleinschreibung verwenden, die für den jeweiligen String geeignet ist.

Variablen in Strings verwenden

Es kann vorkommen, dass Sie innerhalb eines Strings den Wert einer Variablen verwenden möchten, etwa wenn Sie zwei Variablen für den Vor- und den Nachnamen einer Person haben und zur Anzeige des vollständigen Namens kombinieren möchten:

```

first_name = "ada"
last_name = "lovelace"
❶ full_name = f"{first_name} {last_name}"
print(full_name)

```

full_name.py

Um den Wert einer Variablen in einen String einzufügen, stellen Sie den Buchstaben `f` unmittelbar vor das öffnende Anführungszeichen (❶). Die Namen von Variablen, die Sie innerhalb eines Strings verwenden wollen, müssen Sie in geschweifte Klammern einschließen. Bei der Anzeige des Strings ersetzt Python die Variablen durch ihren Wert.

Diese Strings werden als *F-Strings* bezeichnet. Das *F* steht für »Formatierung«, da Python die Strings formatiert, indem es die von geschweiften Klammern umgebenen Variablennamen durch die entsprechenden Werte ersetzt. Die Ausgabe des vorstehenden Codes sieht daher wie folgt aus:

```
ada lovelace
```

Mit F-Strings lässt sich eine Menge anfangen. Beispielsweise können Sie damit vollständige Benachrichtigungen aus den mit Variablen verknüpften Informationen zusammenstellen:

```

first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
❶ print(f"Hello, {full_name.title()}!")

```

Hier wird der vollständige Name in einem Satz zur Begrüßung der Benutzerin verwendet (❶), wobei die Groß- und Kleinschreibung mit der Methode `title()` korrigiert wird. Dieser Code gibt die folgende einfache, aber wohlformatierte Begrüßung aus:

```
Hello, Ada Lovelace!
```

Sie können mithilfe von F-Strings auch eine längere Nachricht zusammenbauen und diese dann einer Variablen zuweisen:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
❶ message = f"Hello, {full_name.title()}!"
❷ print(message)
```

Dieser Code gibt ebenfalls die Meldung "Hello, Ada Lovelace!" aus, weist sie aber auch einer Variablen zu (❶), was den Aufruf von `print()` bei ❷ viel einfacher macht.

Weißraum hinzufügen

Alle nicht druckbaren Zeichen wie Leerzeichen, Tabulatoren und Zeilenumbrüche werden als *Weißraum* bezeichnet. Damit können Sie die Ausgabe so gliedern, dass sie besser lesbar ist.

Um einen Tabulator hinzuzufügen, verwenden Sie wie bei ❶ die Zeichenfolge `\t`:

```
>>> print("Python")
Python
❶ >>> print("\tPython")
Python
```

Wollen Sie in einem String einen Zeilenumbruch vornehmen, verwenden Sie die Zeichenfolge `\n`:

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

Sie können Tabulatoren und Zeilenumbrüche auch kombinieren. Die Zeichenfolge `\n\t` weist Python an, in die nächste Zeile zu wechseln und diese mit einem Tabulator zu beginnen:

```
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
 Python
  C
 JavaScript
```

Zeilenumbrüche und Tabulatoren werden Sie in den nächsten beiden Kapiteln besonders zu schätzen lernen, in denen Sie damit beginnen werden, aus wenigen Codezeilen viele Ausgabezeilen zu produzieren.

Weißraum entfernen

Zusätzlicher Weißraum in Programmen kann irreführend sein. Für Sie als Programmierer mögen `'python'` und `'python '` zwar sehr ähnlich aussehen, aber für ein Programm sind das zwei völlig verschiedene Strings. Python erkennt das zusätzliche Leerzeichen in `'python '` und misst ihm eine Bedeutung bei – sofern Sie nicht ausdrücklich sagen, dass es bedeutungslos ist.

Es ist wichtig, den Weißraum im Auge zu behalten, vor allem, wenn Sie zwei Strings vergleichen, etwa bei der Überprüfung von Benutzernamen bei der Anmeldung an einer Website. Zusätzlicher Weißraum kann aber auch in einfacheren Situationen zu Verwirrung führen. Daher ist es gut, dass Python einfache Möglichkeiten bietet, um überflüssigen Weißraum von Benutzereingaben zu entfernen.

Python ist in der Lage, am rechten und am linken Ende eines Strings nach Weißraum zu suchen. Um jeglichen Weißraum am rechten Ende zu entfernen, verwenden Sie die Methode `rstrip()`.

```

❶ >>> favorite_language = 'python '
❷ >>> favorite_language
'python '
❸ >>> favorite_language.rstrip()
'python'
❹ >>> favorite_language
'python '
```

Der bei ❶ zu `favorite_language` zugewiesene Wert enthält zusätzlichen Weißraum am Ende des Strings. Wenn Sie Python in einer Terminalsitzung nach diesem Wert fragen, können Sie das Leerzeichen am Ende erkennen (❷). Durch die Anwendung der Methode `rstrip()` auf `favorite_language` bei ❸ wird dieses Leerzeichen entfernt – allerdings nur vorübergehend. Wenn Sie den Wert von `favorite_language` erneut abrufen, stellen Sie fest, dass der String immer noch genauso aussieht, wie er gespeichert wurde, nämlich mit dem zusätzlichen Weißraum (❹).

Um den Weißraum endgültig zu entfernen, müssen Sie den bereinigten Wert der Variablen zuweisen:

```

>>> favorite_language = 'python '
❶ >>> favorite_language = favorite_language.rstrip()
>>> favorite_language
'python'
```

Hier entfernen Sie den Weißraum vom rechten Ende des Strings und weisen den resultierenden Wert dann der ursprünglichen Variablen zu, wie bei (❶) gezeigt. Beim Programmieren werden Sie oft einen Variablenwert ändern und anschließend den neuen Wert der Variablen zuweisen. Dadurch können Sie einen Variablenwert im Rahmen der Programmausführung oder in Reaktion auf eine Benutzereingabe ändern.

Natürlich können Sie Weißraum auch vom linken Ende eines Strings oder von beiden Enden zugleich entfernen. Dazu verwenden Sie die Methode `rstrip()` bzw. `strip()`:

```
❶ >>> favorite_language = ' python '  
❷ >>> favorite_language.rstrip()  
    'python'  
❸ >>> favorite_language.lstrip()  
    'python '  
❹ >>> favorite_language.strip()  
    'python'
```

In diesem Beispiel verwenden wir einen Wert mit Weißraum am Anfang und am Ende (❶). Dann entfernen wir den Weißraum von der rechten Seite (❷), von der linken Seite (❸) und von beiden Seiten (❹). Spielen Sie ruhig ein bisschen mit diesen Kürzungsfunktionen herum, um sich mit der Stringbearbeitung vertraut zu machen. In der Praxis werden sie vor allem dazu verwendet, Benutzereingaben zu bereinigen, bevor sie in einem Programm gespeichert werden.

Entfernen von Präfixen

Bei der Arbeit mit Strings besteht eine weitere häufige Aufgabe darin, ein Präfix zu entfernen. Nehmen wir eine URL mit dem üblichen Präfix `https://`. Wir möchten dieses Präfix entfernen, damit wir uns auf den Teil der URL konzentrieren können, den die Benutzer in eine Adressleiste eingeben müssen. So wirds gemacht:

```
>>> nostarch_url = 'https://nostarch.com'  
>>> nostarch_url.removeprefix('https://')  
    'nostarch.com'
```

Geben Sie den Namen der Variablen ein, gefolgt von einem Punkt, und dann die Methode `removeprefix()`. Innerhalb der Klammern geben Sie das Präfix an, das Sie aus der ursprünglichen Zeichenkette entfernen möchten. Wie die Methoden zum Entfernen von Leerzeichen lässt `removeprefix()` die ursprüngliche Zeichenkette unverändert. Wenn Sie den neuen Wert mit dem entfernten Präfix behalten wollen, weisen Sie ihn entweder der ursprünglichen Variable erneut zu oder weisen Sie ihn einer neuen Variablen zu.

```
>>> simple_url = nostarch_url.removeprefix('https://')
```

Wenn Sie eine URL in einer Adressleiste sehen und der Teil `https://` nicht angezeigt wird, verwendet der Browser wahrscheinlich im Hintergrund eine Methode wie `removeprefix()`.

Syntaxfehler bei der Stringverarbeitung vermeiden

Hin und wieder werden Sie einem *Syntaxfehler* begegnen. Dabei kann Python einen Teil des Programms nicht als gültigen Python-Code erkennen. Wenn Sie beispielsweise innerhalb von einfachen Anführungszeichen einen Apostroph verwenden, interpretiert Python alles, was zwischen dem öffnenden Anführungszeichen und dem Apostroph steht, als String und glaubt, dass es sich bei dem Rest um Python-Code handelt, was natürlich zu einem Fehler führt.

Das folgende Beispiel zeigt, wie Sie einfache und doppelte Anführungszeichen korrekt verwenden. Speichern Sie das Programm als *apostrophe.py* und führen Sie es aus:

```
message = "One of Python's strengths is its diverse community." apostrophe.py
print(message)
```

Da der Apostroph hier innerhalb eines von doppelten Anführungszeichen begrenzten Strings steht, kann der Python-Interpreter den String korrekt lesen:

```
One of Python's strengths is its diverse community.
```

Wenn Sie dagegen einfache Anführungszeichen verwenden, kann Python nicht erkennen, wo der String wirklich enden soll:

```
message = 'One of Python's strengths is its diverse community.'
print(message)
```

In diesem Fall erhalten Sie folgende Ausgabe:

```
File "apostrophe.py", line 1
  message = 'One of Python's strengths is its diverse community.'
                                                    ^ ❶
SyntaxError: unterminated string literal (detected at line 1)
```

Diese Ausgabe zeigt, dass bei ❶ unmittelbar hinter dem letzten Anführungszeichen (dem Apostroph) ein Syntaxfehler aufgetreten ist, da der Interpreter einen Teil des Programms nicht als gültigen Python-Code erkannt hat und vermutet, dass das mit einem falsch benannten String zu tun hat. Es gibt verschiedene Ursachen für

Fehler, und ich werde im Laufe unserer Erörterung jeweils auf übliche Fehlerquellen hinweisen. Während Sie noch lernen, Python-Code zu schreiben, werden Sie häufig Syntaxfehlern begegnen. Die Fehlermeldungen dieser Art sind ziemlich allgemein gehalten, sodass es schwierig und frustrierend sein kann, die Fehler aufzuspüren und zu beheben. Wenn Sie bei einem besonders hartnäckigen Fehler nicht weiterkommen, schlagen Sie in Anhang C nach.



Hinweis

Die Syntaxkennzeichnung in Ihrem Editor kann Ihnen helfen, einige Syntaxfehler schneller zu erkennen. Wenn Code in der Farbe für Text oder Text in der Farbe für Code angezeigt wird, dann haben Sie wahrscheinlich irgendwo in der Datei ein Anführungszeichen nicht richtig gesetzt.

Probieren Sie es selbst aus!

Speichern Sie für die folgenden Übungen die einzelnen Programme jeweils in eigenen Dateien mit Namen wie *name_cases.py*. Wenn Sie nicht mehr weiterkommen, legen Sie eine Pause ein oder sehen Sie sich die Vorschläge in Anhang C an.

2-3 Persönliche Nachricht: Weisen Sie den Namen einer Person einer Variablen zu und geben Sie eine einfache Nachricht an diese Person aus, z. B.: »Hello Eric, would you like to learn some Python today?«

2-4 Groß- und Kleinschreibung von Namen: Weisen Sie den Namen einer Person einer Variablen zu und geben Sie den Namen in Kleinbuchstaben, in Großbuchstaben sowie mit großen Anfangsbuchstaben aus.

2-5 Bekanntes Zitat: Geben Sie ein bekanntes Zitat und den Namen von dessen Urheber aus. Die Ausgabe sollte so aussehen wie im folgenden Beispiel, einschließlich der Anführungszeichen:

```
Albert Einstein once said, "A person who never made a mistake never tried anything new."
```

2-6 Bekanntes Zitat 2: Wiederholen Sie Übung 2-5, weisen Sie aber diesmal den Namen des Urhebers der Variablen *famous_person* zu. Bauen Sie die Nachricht zusammen, weisen Sie sie der neuen Variablen *message* zu und geben Sie sie aus.

2-7 Namen bereinigen: Weisen Sie den Namen einer Person einschließlich einiger Weißraumzeichen am Anfang und Ende einer Variablen zu. Verwenden Sie dabei sowohl `\t` als auch `\n` jeweils mindestens einmal.

Geben Sie den Namen einmal einschließlich Weißraum aus und dann dreimal mithilfe der Funktionen `lstrip()`, `rstrip()` und `strip()`.



2-8 Dateierweiterungen: Python verfügt über eine Methode `removesuffix()`, die genau wie `removeprefix()` funktioniert. Weisen Sie den Wert `'python_notes.txt'` einer Variablen namens `filename` zu. Verwenden Sie dann die Methode `removesuffix()`, um den Dateinamen ohne die Dateierweiterung anzuzeigen, wie es einige Dateibrowser tun.

Zahlen

Zahlen werden in der Programmierung sehr häufig verwendet, etwa um Spielstände darzustellen, um Daten zu visualisieren, um Informationen in Webanwendungen zu speichern usw. Je nachdem, wie die Zahlen verwendet werden, behandelt Python sie auf unterschiedliche Weise. Als Erstes sehen wir uns an, wie Python mit ganzen Zahlen (*Integer*) umgeht, da die Arbeit mit ihnen am einfachsten ist.

Integer

In Python können Sie Integer addieren (+), subtrahieren (-), multiplizieren (*) und dividieren (/).

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

In einer Terminalsitzung gibt Python einfach das Ergebnis der Operation zurück. Zur Darstellung von Exponenten verwendet Python zwei Sternchen:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python richtet sich auch nach der mathematischen Reihenfolge von Operationen, sodass Sie mehrere Operationen in einem Ausdruck kombinieren können. Um die Reihenfolge zu ändern, können Sie Klammern angeben:

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

Die Leerzeichen in diesen Beispielen haben keine Auswirkungen darauf, wie Python die Ausdrücke auswertet. Sie ermöglichen es Ihnen nur, beim Lesen schneller zu erkennen, welche Operationen Priorität haben.

Fließkommazahlen

Zahlen mit einem Dezimalpunkt werden in den meisten Programmiersprachen und auch in Python als *Fließkommazahlen* bezeichnet, weil der Dezimalpunkt (der dem Komma in der deutschen Schreibweise entspricht) an beliebiger Stelle stehen kann. Alle Programmiersprachen müssen in der Lage sein, Dezimalzahlen korrekt zu handhaben, unabhängig davon, wo der Dezimalpunkt steht.

Meistens können Sie Dezimalzahlen verwenden, ohne sich Gedanken über das Verhalten zu machen. Geben Sie einfach die gewünschten Zahlen ein, und Python wird sehr wahrscheinlich das tun, was Sie erwarten:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

Es kann jedoch auch vorkommen, dass das Ergebnis eine willkürliche Anzahl von Stellen aufweist:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

Das kann in allen Programmiersprachen geschehen und ist kein Grund zur Besorgnis. Python versucht, das Ergebnis so genau wie möglich wiederzugeben, was aufgrund der Art und Weise, wie die Zahlen auf dem Computer intern dargestellt werden, zu Schwierigkeiten führt. Sie können solche überzähligen Stellen vorläufig ignorieren. Wie Sie bei Bedarf damit umgehen, erfahren Sie in den Projekten in Teil II.

Integer und Fließkommazahlen

Wenn Sie zwei Zahlen dividieren, erhalten Sie immer eine Fließkommazahl, auch wenn es sich bei den beiden Zahlen um Integer handelt und das Ergebnis eine ganze Zahl ist:

```
>>> 4/2
2.0
```

Ebenso erhalten Sie eine Fließkommazahl, wenn Sie in den anderen Operationen Integer und Fließkommazahlen mischen:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

In allen Operationen, an denen Fließkommazahlen beteiligt sind, verwendet Python standardmäßig Fließkommazahlen, auch wenn das Ergebnis eine ganze Zahl ist.

Unterstriche in Zahlen

Wenn Sie sehr große Zahlen schreiben, können Sie der besseren Lesbarkeit halber Unterstriche als Tausendertrennzeichen verwenden:

```
>>> universe_age = 14_000_000_000
```

Bei der Ausgabe zeigt Python aber nur die Ziffern einer solchen mit Unterstrichen definierten Zahl an:

```
>>> print(universe_age)
14000000000
```

Beim Speichern solcher Werte ignoriert Python die Unterstriche. Auch wenn Sie eine Zahl mit den Unterstrichen auf andere Weise aufteilen als in Dreiergruppen, bleibt der Wert unbeeinträchtigt. Für Python sind 1000, 1_000 und 10_00 identisch. Diese Verwendung von Unterstrichen ist sowohl für Integer als auch für Fließkommazahlen möglich.

Mehrfachzuweisung

Sie können innerhalb einer Zeile Code Werte zu mehr als einer Variablen zuweisen. Das macht die Programme kürzer und leichter lesbar. Diese Technik wird vor allem bei der Initialisierung eines zusammengehörigen Satzes von Variablen angewendet.

Beispielsweise können Sie die Variablen *x*, *y* und *z* wie folgt mit 0 initialisieren:

```
>>> x, y, z = 0, 0, 0
```

Die Variablennamen und die Werte müssen Sie jeweils durch Kommata trennen. Python weist dann jeder Variablen den entsprechenden Wert in der Reihenfolge zu. Diese Zuordnung erfolgt korrekt, sofern es gleich viele Variablen wie Werte gibt.

Konstanten

Eine *Konstante* ist eine Variable, deren Wert während der gesamten Lebensdauer des Programms unverändert bleibt. Python hat zwar keinen vordefinierten Typ für Konstanten, allerdings verwenden Python-Programmierer vereinbarungsgemäß Großbuchstaben, um anzuzeigen, dass eine Variable als Konstante behandelt und nie geändert werden soll:

```
MAX_CONNECTIONS = 5000
```

Wenn Sie eine Variable in Ihrem Code als Konstante behandeln wollen, sollten Sie ihren Namen ebenfalls in Großbuchstaben schreiben.

Probieren Sie es selbst aus!

2-9 Alles auf 8: Schreiben Sie eine Addition, eine Subtraktion, eine Multiplikation und eine Division, die jeweils 8 ergeben. Schließen Sie die Operationen in `print`-Anweisungen ein, damit Sie das Ergebnis sehen können. Ihr Programm sollte aus vier Zeilen wie den folgenden bestehen:

```
print(5+3)
```

Die Ausgabe soll vier Zeilen umfassen, in denen jeweils das Ergebnis 8 auf der rechten Seite steht.

2-10 Lieblingszahl: Weisen Sie Ihre Lieblingszahl einer Variablen zu. Stellen Sie mithilfe dieser Variablen eine Nachricht zusammen, in der diese Lieblingszahl angegeben wird, und geben Sie diese Nachricht aus.

Kommentare

In allen Programmiersprachen gibt es die äußerst nützliche Einrichtung von Kommentaren. Alles, was Sie bis jetzt in den Beispielprogrammen geschrieben haben, war Python-Code. Wenn Ihre Programme länger und komplizierter werden, sollten Sie jedoch auch Hinweise hinzufügen, die Ihre Vorgehensweise zur Lösung des Problems beschreiben. In einem *Kommentar* können Sie solche Hinweise in normaler Alltagssprache in Ihre Programme aufnehmen.

Wie werden Kommentare geschrieben?

In Python dient das amerikanische Nummernsymbol (#) zur Kennzeichnung eines Kommentars. Alles, was in einer Zeile auf dieses Zeichen folgt, wird vom Python-Interpreter ignoriert, wie das folgende Beispiel zeigt:

```
# Begrüßt alle mit "Hello".  
print("Hello Python people!")
```

`comment.py`

Hier ignoriert Python die erste Zeile und führt nur die zweite aus:

```
Hello Python people!
```

Was für Kommentare sind sinnvoll?

Der Hauptzweck von Kommentaren besteht darin, zu erklären, was der Code tun soll und wie Sie versuchen, dies zu erreichen. Während Sie an einem Projekt arbeiten, ist Ihnen völlig klar, wie die einzelnen Teile zusammenwirken, aber wenn Sie sich nach einiger Zeit wieder mit einem älteren Projekt beschäftigen, haben Sie wahrscheinlich einige der Einzelheiten vergessen. Sie könnten den Code zwar ausgiebig studieren und herauszufinden versuchen, wie die Einzelteile zusammenwirken, aber wenn Sie Ihre Vorgehensweise in Form von guten, verständlichen Kommentaren erklärt haben, hilft Ihnen das, viel Zeit zu sparen.

Wenn Sie Programmierung als Beruf anstreben oder mit anderen Programmierern zusammenarbeiten wollen, müssen Sie aussagekräftige Kommentare schreiben. Heute wird die meiste Software im Team geschrieben. Das kann eine Gruppe von Angestellten einer Firma sein, aber auch eine Gruppe von Personen, die gemeinsam an einem Open-Source-Projekt arbeiten. Erfahrene Programmierer erwarten, Kommentare im Code zu sehen, weshalb Sie es sich am besten gleich angewöhnen sollten, beschreibende Kommentare in Ihre Programme aufzunehmen. Klare und prägnante Kommentare zu schreiben gehört zu den nützlichsten Dingen, die Sie sich als angehender Programmierer angewöhnen können.

Wenn Sie an einer Stelle überlegen, ob Sie einen Kommentar einfügen sollten oder nicht, fragen Sie sich, ob Sie mehrere verschiedene Ansätze erwogen haben, bevor Sie auf eine sinnvolle Vorgehensweise gestoßen sind. Wenn das der Fall ist, dann beschreiben Sie Ihre Lösung in einem Kommentar. Es ist viel einfacher, später überflüssige Kommentare zu löschen, als ein unzureichend kommentiertes Programm nachträglich mit Kommentaren zu versehen. Von jetzt an werde ich auch in den Beispielen in diesem Buch Kommentare angeben, um einzelne Abschnitte des Codes zu erklären.

Probieren Sie es selbst aus!

2-11 Kommentare hinzufügen: Wählen Sie zwei Ihrer bisher geschriebenen Programme aus und fügen Sie in jedes mindestens einen Kommentar ein. Wenn Sie nicht wissen, was Sie schreiben sollen, da die bisherigen Programme extrem einfach waren, fügen Sie einfach am Anfang der Programmdatei Ihren Namen und das Datum ein und geben Sie in einem Satz an, was das Programm macht.

The Zen of Python

Erfahrene Python-Programmierer ermuntern Sie dazu, Komplexität zu vermeiden und nach Möglichkeit Einfachheit anzustreben. Die Grundhaltung der Python-Community kommt in dem Text »The Zen of Python« von Tim Peters zum Ausdruck. Diese knappe Aufstellung von Prinzipien zum Schreiben von gutem Python-Code können Sie sich ansehen, indem Sie im Interpreter `import this` eingeben.

```
>>> import this
The Zen of Python, by Tim Peters
```

Ich werde »The Zen of Python« hier nicht komplett wiedergeben, sondern nur einige Zeilen daraus anführen, um Ihnen zu zeigen, warum diese Prinzipien für Sie als angehender Python-Programmierer so wichtig sind.

```
Beautiful is better than ugly.
```

Für Python-Programmierer kann Code schön und elegant sein. Bei der Programmierung geht es darum, Probleme zu lösen. Programmierer haben schon immer gut gestaltete, effiziente und auch schöne Lösungen für Probleme gewürdigt. Wenn Sie mehr über Python lernen und mehr Code schreiben, wird Ihnen eines Tages jemand über die Schulter schauen und sagen: »Wow, das ist wirklich schöner Code!«

Einfach ist besser als komplex.

Wenn Sie die Wahl zwischen einer einfachen und einer komplexen Lösung haben, die beide funktionieren, dann entscheiden Sie sich für die einfache. Ihr Code lässt sich dann einfacher pflegen, und Sie und andere können später einfacher darauf aufbauen.

Komplex ist besser als kompliziert.

Das Leben ist nicht immer einfach, und manchmal gibt es keine einfache Lösung für ein Problem. Nehmen Sie in einem solchen Fall die einfachste funktionierende Lösung.

Lesbarkeit zählt.

Selbst wenn Ihr Code komplex ist, sollten Sie stets darauf achten, ihn lesbar zu gestalten. Nehmen Sie informative Kommentare in den Code auf.

Es sollte eine `--` und zwar vorzugsweise nur eine `--` offensichtliche Möglichkeit geben.

Wenn zwei Python-Programmierer gebeten werden, dasselbe Problem zu lösen, sollten dabei ziemlich ähnliche Lösungen herauskommen. Das heißt jedoch nicht, dass es in der Programmierung keinen Platz für Kreativität gäbe. Ganz im Gegenteil! Aber Programmierung besteht zum großen Teil aus kleinen, gängigen Lösungsansätzen für einfache Lösungen im Rahmen größerer Projekte, die mehr Kreativität erfordern. Die grundlegenden Mechanismen Ihrer Programme sollten auch für andere Python-Programmierer sofort einsichtig sein.

Jetzt ist besser als nie.

Sie können Ihr ganzes Leben damit zubringen, alle Feinheiten von Python und der Programmierung im Allgemeinen zu lernen, aber dann würden Sie niemals dazu kommen, ein Projekt fertigzustellen. Versuchen Sie nicht, perfekten Code zu schreiben, sondern schreiben Sie Code, der funktioniert. Dann können Sie immer noch entscheiden, ob Sie den Code verbessern oder mit einem neuen Projekt anfangen wollen.

Behalten Sie diese Prinzipien von Einfachheit und Klarheit im Hinterkopf, wenn Sie die nächsten Kapitel durcharbeiten und in die anspruchsvolleren Themen einsteigen. Erfahrene Programmierer werden Ihren Code dann stärker würdigen, Ihnen Rückmeldung geben und gern an interessanten Projekten mit Ihnen zusammenarbeiten.

Probieren Sie es selbst aus!

2-12 The Zen of Python: Geben Sie in einer Python-Terminalsitzung `import this` ein und schauen Sie sich auch die anderen aufgeführten Prinzipien an.

Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie Sie mit Variablen arbeiten. Sie haben erfahren, dass Sie beschreibende Variablennamen verwenden sollten, und es wurde Ihnen gezeigt, wie Sie Namens- und Syntaxfehler beheben können. Außerdem ging es darum, was Strings sind und wie Sie sie in Kleinbuchstaben, in Großbuchstaben und mit großem Anfangsbuchstaben anzeigen können. Des Weiteren haben Sie gelernt, wie Sie unnötige Elemente aus verschiedenen Teilen eines Strings entfernen, um Ausgaben sauber zu formatieren. Sie haben auch schon Integer und Fließkommazahlen verwendet und einige Möglichkeiten für die Arbeit mit numerischen Daten kennengelernt. Zudem haben Sie erfahren, wie Sie erklärende Kommentare schreiben, um Ihren Code für sich selbst und andere leichter verständlich zu machen. Am Ende des Kapitels wurde Ihnen das Grundprinzip vorgestellt, Ihren Code so einfach wie möglich zu halten.

In Kapitel 3 erfahren Sie, wie Sie Gruppen von Informationen in besonderen Datenstrukturen, sogenannten *Listen*, speichern können und wie Sie eine solche Liste durcharbeiten und die darin enthaltenen Informationen bearbeiten.

Inhalt

Der Autor	xxiv
Der Fachgutachter	xxiv
Danksagung	xxv
Vorwort zur dritten Auflage	xxvii
Einleitung	xxxI
Zielgruppe	xxxii
Lernstoff	xxxii
Onlinematerial	xxxiv
Warum Python?	xxxiv
Teil 1 Grundlagen	1
1 Erste Schritte	3
Die Programmierumgebung einrichten	3
Python-Versionen	4
Python-Codeausschnitte ausführen	4
Der Editor VS Code	5
Python auf verschiedenen Betriebssystemen	5
Python unter Windows	5
Python unter macOS	7
Python unter Linux	9
Das Hello-World-Programm ausführen	11
Die Python-Erweiterung für VS Code installieren	11
Hello_world.py ausführen	11
Fehlersuche	12

Python-Programme im Terminal ausführen	14
Unter Windows	14
Unter Linux und macOS	15
Zusammenfassung	16
2 Variablen und einfache Datentypen	17
Was bei der Ausführung von hello_world.py wirklich geschieht	17
Variablen	18
Variablen benennen und verwenden	19
Fehler bei Variablennamen vermeiden	20
Variablen sind Etiketten	21
Strings	22
Groß- und Kleinschreibung mithilfe von Methoden ändern	23
Variablen in Strings verwenden	24
Weißraum hinzufügen	25
Weißraum entfernen	26
Entfernen von Präfixen	27
Syntaxfehler bei der Stringverarbeitung vermeiden	28
Zahlen	30
Integer	30
Fließkommazahlen	31
Integer und Fließkommazahlen	32
Unterstriche in Zahlen	32
Mehrfachzuweisung	33
Konstanten	33
Kommentare	34
Wie werden Kommentare geschrieben?	34
Was für Kommentare sind sinnvoll?	34
The Zen of Python	35
Zusammenfassung	37
3 Eine Einführung in Listen	39
Was sind Listen?	39
Elemente in einer Liste ansprechen	40
Indizes beginnen bei 0, nicht bei 1	41
Einzelne Werte aus einer Liste verwenden	41

Elemente verändern, hinzufügen und entfernen	42
Elemente in einer Liste ändern	43
Elemente zu einer Liste hinzufügen	43
Elemente aus einer Liste entfernen	45
Listen ordnen	50
Listen mit <code>sort()</code> dauerhaft sortieren	50
Listen mit der Funktion <code>sorted()</code> vorübergehend sortieren	50
Listen in umgekehrter Reihenfolge ausgeben	51
Die Länge einer Liste ermitteln	52
Indexfehler vermeiden	53
Zusammenfassung	55
4 Mit Listen arbeiten	57
Eine komplette Liste durchlaufen	57
Die Schleife im Detail	58
Weitere Aufgaben in einer <code>for</code> -Schleife erledigen	59
Aktionen nach der <code>for</code> -Schleife	61
Einrückungsfehler vermeiden	62
Vergessene Einrückung der ersten Zeile in einer Schleife	62
Vergessene Einrückung nachfolgender Zeilen	63
Unnötige Einrückung	63
Unnötige Einrückung nach einer Schleife	64
Vergessener Doppelpunkt	65
Numerische Listen	66
Die Funktion <code>range()</code>	66
Numerische Listen mithilfe von <code>range()</code> aufstellen	67
Einfache Statistiken für numerische Listen	69
Listennotation	69
Teillisten	71
Einen Slice erstellen	71
Einen Slice in einer Schleife durchlaufen	73
Listen kopieren	74
Tupel	77
Ein Tupel definieren	77
Die Werte in einem Tupel durchlaufen	78
Tupel überschreiben	78

Code formatieren	79
Die Gestaltungsrichtlinien	80
Einrückung	80
Zeilenlänge	80
Leerzeilen	81
Zusammenfassung	82
5 if-Anweisungen	83
Ein einfaches Beispiel	83
Bedingungen	84
Prüfung auf Gleichheit	84
Groß- und Kleinschreibung bei der Prüfung auf Gleichheit	85
Prüfung auf Ungleichheit	86
Numerische Vergleiche	87
Prüfung auf mehrere Bedingungen	87
Prüfung auf Vorhandensein eines Werts in einer Liste	89
Prüfung auf Abwesenheit eines Werts in einer Liste	89
Boolesche Ausdrücke	90
if-Anweisungen	91
Einfache if-Anweisungen	91
if-else-Anweisungen	92
Die if-elif-else-Kette	93
Mehrere elif-Blöcke	95
Den else-Block weglassen	95
Mehrere Bedingungen prüfen	96
if-Anweisungen für Listen	99
Prüfung auf besondere Elemente	99
Prüfung auf nicht leere Liste	100
Mehrere Listen verwenden	101
if-Anweisungen gestalten	103
Zusammenfassung	104
6 Dictionaries	105
Ein einfaches Dictionary	106
Umgang mit Dictionaries	106
Zugriff auf die Werte in einem Dictionary	107
Schlüssel-Wert-Paare hinzufügen	108
Ein leeres Dictionary als Ausgangspunkt	108

Werte in einem Dictionary ändern	109
Schlüssel-Wert-Paare entfernen	111
Ein Dictionary aus ähnlichen Objekten	111
Mit get() auf Werte zugreifen	113
Dictionaries in einer Schleife durchlaufen	114
Alle Schlüssel-Wert-Paare durchlaufen	115
Alle Schlüssel in einem Dictionary durchlaufen	117
Die Schlüssel in einem Dictionary geordnet durchlaufen	119
Alle Werte in einem Dictionary durchlaufen	119
Verschachtelung	122
Dictionaries in einer Liste	122
Listen in einem Dictionary	125
Dictionaries in einem Dictionary	127
7 Benutzereingaben und while-Schleifen	131
Die Funktion input()	132
Klar verständliche Eingabeaufforderungen schreiben	132
Verwendung von int() für numerische Eingaben	133
Der Modulo-Operator	135
while-Schleifen	136
while-Schleifen in Aktion	136
Programmbeendigung durch den Benutzer	137
Flags	139
Eine Schleife mit break verlassen	140
Die Anweisung continue	141
Endlosschleifen vermeiden	142
while-Schleifen für Listen und Dictionaries	143
Elemente von einer Liste in eine andere verschieben	144
Alle Vorkommen eines Wertes aus einer Liste entfernen	145
Ein Dictionary mit Benutzereingaben füllen	145
Zusammenfassung	147
8 Funktionen	149
Funktionen definieren	150
Informationen an eine Funktion übergeben	150
Argumente und Parameter	151
Argumente übergeben	152
Positionsabhängige Argumente	152
Schlüsselwortargumente	154

Standardwerte	155
Verschiedene Formen für Funktionsaufrufe	156
Argumentfehler vermeiden	157
Rückgabewerte	158
Einen einfachen Wert zurückgeben	158
Optionale Argumente	159
Ein Dictionary zurückgeben	161
Funktionen in einer while-Schleife	162
Eine Liste übergeben	164
Eine Liste mithilfe einer Funktion ändern	165
Die Änderung einer Liste in einer Funktion verhindern	168
Beliebig viele Argumente übergeben	169
Positionsabhängige Argumente und Argumente beliebiger Anzahl kombinieren	170
Beliebig viele Schlüsselwortargumente übergeben	171
Funktionen in Modulen speichern	173
Ein komplettes Modul importieren	173
Einzelne Funktionen importieren	174
Eine Funktion mit »as« umbenennen	175
Ein Modul mit »as« umbenennen	176
Alle Funktionen eines Moduls importieren	176
Gestaltung von Funktionen	177
Zusammenfassung	178
9 Klassen	181
Eine Klasse erstellen und verwenden	182
Die Klasse Dog erstellen	182
Eine Instanz einer Klasse anlegen	184
Mit Klassen und Instanzen arbeiten	187
Die Klasse Car	187
Einen Standardwert für ein Attribut festlegen	188
Attributwerte bearbeiten	189
Vererbung	193
Die Methode <code>__init__()</code> für eine Kindklasse	193
Attribute und Methoden der Kindklasse definieren	195
Methoden der Elternklasse überschreiben	196
Instanzen als Attribute	197
Reale Objekte modellieren	199

Klassen importieren	200
Eine einzelne Klasse importieren	201
Mehrere Klassen in einem Modul speichern	202
Mehrere Klassen aus einem Modul importieren	204
Ein gesamtes Modul importieren	204
Alle Klassen eines Moduls importieren	205
Ein Modul in ein Modul importieren	205
Aliase verwenden	206
Ihren eigenen Arbeitsablauf finden	207
Die Standardbibliothek von Python	208
Gestaltung von Klassen	209
Zusammenfassung	210
10 Dateien und Ausnahmen	211
Aus Dateien lesen	212
Den Inhalt einer Datei lesen	212
Relative und absolute Dateipfade	214
Lesen der Zeilen einer Datei	215
Dateiinhalte verarbeiten	216
Große Dateien: eine Million Stellen	217
Ist Ihr Geburtsdatum in Pi enthalten?	218
In Dateien schreiben	220
Eine einzelne Zeile schreiben	220
Mehrere Zeilen schreiben	220
Ausnahmen	222
Division durch null	222
try-except-Blöcke	223
Abstürze mithilfe von Ausnahmen verhindern	224
Der else-Block	225
Datei nicht gefunden	226
Text analysieren	227
Umgang mit mehreren Dateien	228
Fehler stillschweigend übergehen	230
Welche Fehler sollten Sie melden und welche nicht?	231
Daten speichern	232
json.dump() und json.load()	233
Benutzergenerierte Daten speichern und lesen	234
Refactoring	236
Zusammenfassung	239

11 Code testen	241
Pytest mit pip installieren	242
pip aktualisieren	242
Pytest installieren	243
Funktionen testen	244
Unit-Tests und Testfälle	245
Ein bestandener Test	245
Ein nicht bestandener Test	247
Was tun bei einem nicht bestandenen Test?	248
Neue Tests hinzufügen	249
Klassen testen	251
Verschiedene Annahmen	251
Eine Beispielklasse zum Testen	252
Die Klasse AnonymousSurvey testen	254
Verwenden von Fixtures	256
Zusammenfassung	258
Teil 2 Projekte	261
Alien Invasion – ein Python-Spiel	261
Datenvisualisierung	262
Webanwendungen	262
Projekt 1: Alien Invasion	263
12 Das eigene Kampfschiff	265
Das Projekt planen	266
Pygame installieren	267
Erste Schritte für das Spielprojekt	267
Ein Pygame-Fenster anlegen und auf Benutzereingaben reagieren	267
Steuerung der Framerate	269
Die Hintergrundfarbe festlegen	270
Eine Klasse für Einstellungen anlegen	271
Das Bild eines Raumschiffs hinzufügen	272
Die Klasse Ship	273
Das Schiff auf den Bildschirm zeichnen	275

Refactoring: Die Methoden <code>_check_events()</code> und <code>_update_screen()</code>	277
Die Methode <code>_check_events()</code>	277
Die Methode <code>_update_screen()</code>	278
Das Schiff bewegen	279
Auf Tastenbetätigungen reagieren	279
Kontinuierliche Bewegung	280
Bewegung nach rechts und links	282
Die Geschwindigkeit des Schiffes anpassen	283
Den Bewegungsbereich des Schiffes einschränken	285
Refactoring von <code>_check_events()</code>	286
Beenden mit Q	286
Das Spiel im Vollbildmodus ausführen	287
Zwischenstand	288
<code>alien_invasion.py</code>	288
<code>settings.py</code>	288
<code>ship.py</code>	289
Geschosse	289
Einstellungen für Geschosse hinzufügen	289
Die Klasse <code>Bullet</code>	290
Geschosse in Gruppen speichern	291
Geschosse abfeuern	292
Alte Geschosse löschen	294
Die Anzahl der Geschosse begrenzen	295
Die Methode <code>_update_bullets()</code>	296
Zusammenfassung	297
13 Die Außerirdischen	299
Überblick über das Projekt	300
Das erste Invasionsschiff	300
Die Klasse <code>Alien</code>	301
Eine Instanz von <code>Alien</code> erstellen	302
Die Invasionsflotte erstellen	304
Reihen von Invasionsschiffen erstellen	304
Refactoring von <code>_create_fleet()</code>	306
Reihen hinzufügen	306

Die Flotte in Bewegung setzen	309
Die Invasoren nach rechts bewegen	309
Einstellungen für die Flugrichtung der Flotte	311
Auf Randberührungen prüfen	311
Sinken und Flugrichtung ändern	312
Invasoren abschießen	313
Kollisionen von Geschossen erkennen	314
Größere Geschosse zu Testzwecken	315
Die Flotte auffüllen	316
Die Geschosse beschleunigen	317
Refactoring von <code>_update_bullets()</code>	317
Spielende	318
Kollisionen zwischen Invasoren und dem eigenen Schiff erkennen . . .	318
Auf Kollisionen zwischen Invasoren und dem eigenen Schiff reagieren	319
Wenn Invasoren den unteren Bildschirmrand erreichen	322
Game over!	323
Welche Teile des Spiels müssen ausgeführt werden?	324
Zusammenfassung	325
14 Das Wertungssystem	327
Eine Play-Schaltfläche hinzufügen	327
Die Klasse <code>Button</code>	328
Die Schaltfläche auf den Bildschirm zeichnen	330
Das Spiel starten	331
Das Spiel zurücksetzen	332
Die Play-Schaltfläche deaktivieren	333
Den Mauszeiger ausblenden	333
Levels	334
Die Geschwindigkeitseinstellungen ändern	335
Die Geschwindigkeit zurücksetzen	336
Die Punktwertung	337
Den Punktestand anzeigen	338
Eine Anzeigetafel erstellen	339
Den Punktestand bei jedem Abschuss erhöhen	341
Den Punktestand zurücksetzen	341

Alle Treffer berücksichtigen	342
Den Punktwert erhöhen	343
Den Punktestand runden	344
Highscore	345
Das Level anzeigen	347
Die Anzahl der verfügbaren Schiffe anzeigen	350
Zusammenfassung	353

Projekt 2: Datenvisualisierung 355

15 Daten generieren 357

Matplotlib installieren	358
Einfache Liniendiagramme	359
Beschriftung und Linienstärke ändern	360
Das Diagramm korrigieren	361
Vordefinierte Formatierungen verwenden	363
Einzelne Punkte mit scatter() darstellen und gestalten	364
Eine Folge von Punkten mit scatter() ausgeben	365
Daten automatisch berechnen	366
Eigene Beschriftungen anlegen	368
Eigene Farben festlegen	368
Eine Colormap verwenden	368
Diagramme automatisch speichern	370
Zufallsbewegungen	370
Die Klasse RandomWalk	371
Richtungen wählen	371
Den Zufallspfad als Diagramm ausgeben	373
Mehrere Zufallspfade erstellen	374
Den Pfad gestalten	375
Würfeln mit Plotly	380
Plotly installieren	381
Die Klasse <i>Die</i>	381
Würfeln	382
Die Ergebnisse analysieren	382
Ein Histogramm erstellen	383
Das Diagramm anpassen	385
Ergebnisse bei zwei Würfeln	386

Weitere Anpassungen	387
Würfel unterschiedlicher Flächenzahl	388
Diagramme speichern	389
Zusammenfassung	390
16 Daten herunterladen	391
Das Dateiformat CSV	392
CSV-Spaltenköpfe analysieren	392
Die Spaltenköpfe und ihre Position ausgeben	394
Daten entnehmen und lesen	394
Daten in einem Temperaturdiagramm darstellen	395
Das Modul datetime	396
Datumsangaben im Diagramm darstellen	398
Ein Diagramm für einen längeren Zeitraum	399
Eine zweite Datenreihe darstellen	400
Einen Diagrammbereich einfärben	401
Fehlerprüfung	402
Daten selbst herunterladen	406
Globale Daten im GeoJSON-Format visualisieren	407
Erdbebendaten herunterladen	408
JSON-Daten untersuchen	408
Eine Liste aller Erdbeben aufstellen	411
Die Stärken entnehmen	411
Ortsdaten entnehmen	412
Eine Weltkarte zeichnen	413
Darstellung der Stärke	414
Die Farben der Markierungen anpassen	415
Weitere Farbpaletten	417
Maustext hinzufügen	417
Zusammenfassung	419
17 APIs	421
APIs	421
Git und GitHub	422
Daten mithilfe eines API-Aufrufs anfordern	422
Das Paket requests installieren	423
API-Antworten verarbeiten	424
Das Antwort-Dictionary verarbeiten	425

Ein Überblick über die höchstbewerteten Repositories	428
Grenzwerte für die API-Aufruftrate	429
Angaben zu Repositories mit Plotly visualisieren	430
Das Diagramm gestalten	431
Eigenen Maustext hinzufügen	432
Links zu dem Diagramm hinzufügen	434
Marker-Farben anpassen	435
Mehr über Plotly und die GitHub-API	436
Die API von Hacker News	436
Zusammenfassung	440
Projekt 3: Webanwendungen	443
18 Erste Schritte mit Django	445
Ein Projekt einrichten	446
Eine Spezifikation schreiben	446
Eine virtuelle Umgebung erstellen	447
Die virtuelle Umgebung aktivieren	447
Django installieren	448
Ein Projekt in Django erstellen	449
Die Datenbank erstellen	449
Das Projekt anzeigen	450
Eine App anlegen	452
Modelle definieren	453
Modelle aktivieren	454
Die Admin-Site von Django	455
Das Modell für die Einträge definieren	459
Das Modell Entry in die Datenbank aufnehmen	460
Das Modell Entry auf der Admin-Site registrieren	460
Die Django-Shell	461
Seiten erstellen: die Startseite von Learning Log	464
Eine URL zuordnen	465
Eine Ansicht schreiben	467
Eine Vorlage schreiben	467
Weitere Seiten erstellen	469
Vererbung bei Vorlagen	469
Die Seite Topics	472
Einzelne Fachgebetsseiten	475
Zusammenfassung	479

19 Benutzerkonten	481
Dateneingabe durch die Benutzer	482
Neue Fachgebiete hinzufügen	482
Neue Einträge hinzufügen	487
Einträge bearbeiten	491
Benutzerkonten einrichten	495
Die App accounts	495
Die Anmeldeseite	496
Abmelden	500
Die Registrierungsseite	501
Die Benutzer als Besitzer ihrer eigenen Daten	504
Den Zugriff mit @login_required beschränken	504
Daten mit Benutzern verknüpfen	506
Den Zugriff auf die Fachgebiete auf die zuständigen Benutzer einschränken	510
Die Fachgebiete eines Benutzers schützen	511
Die Seite edit_entry schützen	512
Neue Fachgebiete dem aktuellen Benutzer zuordnen	512
Zusammenfassung	513
20 Eine App gestalten und bereitstellen	515
Learning Log gestalten	516
Die App django-bootstrap5	516
Learning Log mit Bootstrap gestalten	517
Änderungen an base.html	518
Die Startseite mit einem Jumbotron gestalten	524
Das Anmeldeformular gestalten	525
Die Seite Topics gestalten	527
Einträge auf den Fachgebichtsseiten gestalten	528
Learning Log bereitstellen	530
Ein Platform.sh-Konto anlegen	530
Die Platform.sh-Befehlszeile installieren	530
Platformsshconfig installieren	531
Die Datei requirements.txt erstellen	531
Zusätzliche Voraussetzungen für die Bereitstellung	532
Konfigurationsdateien hinzufügen	532
Die Konfigurationsdatei services-yaml	535

settings-py für Platform.sh anpassen	536
Mit Git den Überblick über die Projektdateien bewahren	537
Die Platform.sh-Bereitstellung verbessern	542
Das Onlineprojekt schützen	544
Änderungen mit Commit bestätigen und übertragen	545
Eigene Fehlerseiten erstellen	546
Weiterentwicklung des Projekts	548
Projekte auf Platform.sh löschen	548
Zusammenfassung	550

Nachwort	551
---------------------------	------------

Anhang	553
-------------------------	------------

A Installation und Fehlerbehebung	553
----------------------------------------------------	------------

Python unter Windows	553
Py statt Python	554
Das Installationsprogramm erneut ausführen	554
Python unter macOS	555
Versehentliches Installieren der Apple-Version von Python	555
Python 2 und ältere Versionen von macOS	555
Python unter Linux	555
Die Standardinstallation vornehmen	556
Die letzte Python-Version installieren	556
Die verwendete Python-Version herausfinden	557
Schlüsselwörter und integrierte Funktionen	557
Python-Schlüsselwörter	557
Integrierte Python-Funktionen	558

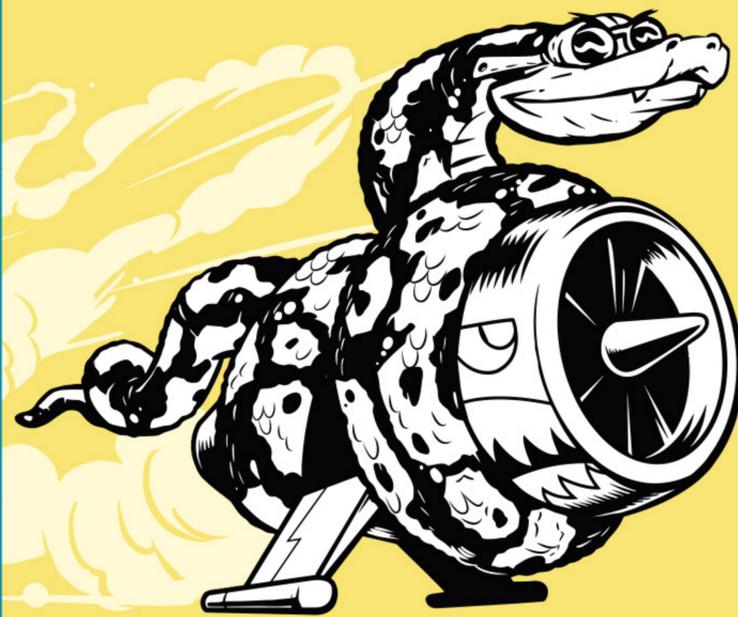
B Texteditoren und IDEs	559
------------------------------------------	------------

Effizient mit VS Code arbeiten	561
VS Code konfigurieren	561
Tabulatoren und Leerzeichen verwenden	561
Das Farbschema ändern	562
Den Zeilenlängenmarker festlegen	562
Die Ausgabe vereinfachen	562
Kurzbefehle in VS Code	564

Codeblöcke einrücken und Einrückungen aufheben	564
Codeblöcke auskommentieren	564
Weitere Texteditoren und IDEs	565
IDLE	565
Geany	566
Sublime Text	566
Emacs und Vim	566
PyCharm	567
Jupyter Notebooks	567
C Hilfe finden	569
Erste Schritte	569
Versuchen Sie es erneut	570
Legen Sie eine Pause ein	570
Nutzen Sie das Onlinematerial zu diesem Buch	571
Online nach Hilfe suchen	571
Stack Overflow	571
Die offizielle Python-Dokumentation	572
Offizielle Dokumentation der Bibliotheken	572
r/learnpython	573
Blogs	573
Discord	573
Slack	573
D Versionssteuerung mit Git	575
Git installieren	576
Git konfigurieren	576
Ein Projekt anlegen	577
Dateien ignorieren	577
Ein Repository initialisieren	577
Den Projektstatus überprüfen	578
Dateien zum Repository hinzufügen	578
Einen Commit durchführen	579
Das Protokoll einsehen	580
Der zweite Commit	580
Änderungen zurücknehmen	581
Vorherige Commits auschecken	583
Das Repository löschen	584

E Versionssteuerung mit Git	587
Git installieren	587
Grundlegende Fehlersuche	589
Den Hinweisen nachgehen	589
Das Protokoll analysieren	590
Betriebssystemspezifische Fehlersuche	592
Von Windows bereitstellen	593
Weitere Entwicklungsansätze	596
 Stichwortverzeichnis	 599

WELTWEITER
BESTSELLER



PYTHON
SCHNELL
LERNEN

»Python Crashkurs« zählt zu den weltweiten Bestsellern unter den Einführungen in die Programmiersprache Python. Mit dieser kompakten und gründlichen Anleitung werden Sie in kürzester Zeit Programme schreiben, Probleme lösen und funktionierende Anwendungen entwickeln.

Zu Beginn führt Sie Autor Eric Matthes in grundlegende Programmierkonzepte wie Variablen, Listen, Klassen und Schleifen ein. Sie werden Ihr neues Wissen in die Praxis umsetzen, indem Sie ein von »Space Invaders« inspiriertes Arcade-Spiel konzipieren und eine Reihe von Datenvisualisierungen mit den praktischen Python-Bibliotheken programmieren.

Mit den Übungen des Buches lernen Sie

- leistungsfähige Python-Bibliotheken und -Werkzeuge zu verwenden, darunter pytest, Pygame, Matplotlib, Plotly und Django
- immer komplexere 2D-Spiele zu erstellen, die auf Tastendruck und Mausklicks reagieren
- Anwendungen zu bauen, mit denen Benutzer Konten erstellen und ihre Daten verwalten können. Stellen Sie Anwendungen online bereit.

- interaktive Datenvisualisierungen mit einer Vielzahl von Datensätzen zu generieren
- Programmierfehler zu beheben und klassische Programmierprobleme zu lösen

Neu in dieser Auflage: Diese 3. Auflage wurde komplett auf den jüngsten Stand des Python-Codes gebracht. Zu den neuen und aktualisierten Themen gehören VS Code für die Textbearbeitung, das pathlib-Modul für die Dateiverwaltung, pytest für das Testen Ihres Codes sowie die neuesten Funktionen von Matplotlib, Plotly und Django.

Wenn Sie in die Programmierung einsteigen wollen, wird Ihnen »Python Crashkurs« die Fähigkeiten vermitteln, schnell echte Programme zu schreiben.

ÜBER DEN AUTOR

Eric Matthes unterrichtete 25 Jahre lang Mathematik und Naturwissenschaften an einer High School und gab Python-Einführungskurse. Matthes ist an einer Reihe von Open-Source-Projekten beteiligt und arbeitet hauptberuflich als Autor und Programmierer. Der Outdoor-Fan lebt im Südosten Alaskas.

Behandelt Python 3.x

plus+ Interesse am E-Book?
www.dpunkt.plus



Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben



ISBN 978-3-86490-989-4

€ 32,90 (D)