Python mit Biss

Eine Einführung in die Programmierung in Python

Michael Hartl

Pearson

Zugangscode

Falls Sie beim Kauf Ihres eBooks keinen Zugangscode erhalten haben, kontaktieren Sie uns bitte über die folgende Seite und halten Sie Ihre Rechnung/Bestellbestätigung bereit:
https://www.pearson.de/ebook-zugangscode

Wir können uns die Schlüssel und Werte getrennt ansehen, die (ab Python 3.6) der *Reihe nach* in speziellen Python-Objekten gespeichert werden:

```
>>> moonwalks.keys()
dict_keys(['Neil Armstrong', 'Buzz Aldrin', 'Alan Shepard',
'Eugene Cernan', 'Michael Jackson'])
>>> moonwalks.values()
dict_values([1969, 1969, 1971, 1972, 1983])
```

Beachten Sie, dass frühere Versionen von Python die Dictionary-Elemente nicht geordnet haben, sodass Sie vorsichtig sein sollten, wenn Sie Annahmen über die Reihenfolge treffen.

Wie ein Listenindex ist ein Schlüssel in einem Dictionary jeweils nur einem bestimmten Wert zugeordnet. Das bedeutet, dass wir den zu einem Schlüssel gehörenden Wert durch einen anderen Wert ersetzen können. Es dürfen auch zwei oder mehr identische Werte in einem Dictionary vorkommen, wie man im obigen Beispiel von `moonwalks` sieht. Ein Dictionary kann aber nicht zwei identische Schlüssel enthalten. Daher ist es manchmal nützlich, sich die Schlüssel eines Dictionary wie ein geordnetes Set vorzustellen, da sie (wie Sets) keine sich wiederholenden Elemente aufweisen können. In der Tat kann das oben erwähnte spezielle `keys()`-Objekt, das technisch als *View* bekannt ist, in manchen Zusammenhängen wie ein Set behandelt werden. Der folgende Code führt beispielsweise die Bestimmung einer Schnittmenge wie in ▶ Abschnitt 3.6 beschrieben durch:

```
>>> apollo_11 = {"Neil Armstrong", "Buzz Aldrin"}
>>> moonwalks.keys() & apollo_11
{'Neil Armstrong', 'Buzz Aldrin'}
```

PYTHON MIT BISS

Dictionaries (Wörterbücher)

Dictionaries (häufig einfach kurz *Dicts* genannt) sind ein sehr praktisches Werkzeug in der Softwareentwicklung. Sie werden nicht unbedingt gebraucht, können gewisse Probleme jedoch sehr vereinfachen. Aber beginnen wir von vorn:

```
a = 1 mein_dict = {"a": 1}
```

Worin liegen die Ähnlichkeiten und die Unterschiede dieser beiden Codezeilen?

- Sie definieren beide ein „Objekt", auf welches mit einem „Schlüssel" `a` zugegriffen werden kann. Der Wert, der dem Schlüssel zugewiesen wird, ist 1. In der ersten Zeile wird zu diesem Zweck eine normale Variable und in der zweiten Zeile ein Dictionary verwendet.
- Einer Variablen muss bei der Deklaration ein Wert zugewiesen werden. Der Wert kann aber zur Laufzeit verändert werden. Ein Dictionary kann anfänglich als leeres Dictionary deklariert und sein Inhalt erst zur Laufzeit z. B. aus einer Datei eingelesen werden.
- Dictionaries haben vielseitige Anwendungen. Sie können beispielsweise häufig if-Anweisungen ersetzen.
- Bei der Verwendung von Dictionaries sollten Sie immer überlegen, ob ein Dictionary im vorliegenden Fall die beste Wahl ist. Zur Strukturierung von Daten sind beispielsweise Klassen (▶ Kapitel 7) häufig besser geeignet.
- Verwenden Sie generell normale Variablen, behalten Sie Dictionaries und Klassen aber immer als zusätzliche Option im Hinterkopf, wenn Sie ein neues Problem lösen müssen.

Übrigens können wie bei Listen (▶ Abschnitt 3.4.1) mit dem Schlüsselwort `in` prüfen, ob ein bestimmter Schlüssel in einem Dictionary vorkommt:

```
>>> "Buzz Aldrin" in moonwalks
True
```

Beachten Sie, dass wir hier `keys()` weglassen und `in` einfach mit dem Namen des gesamten Dictionary verwenden können, um die Schlüssel zu durchsuchen (die Werte werden dabei nicht berücksichtigt). Einem weiteren Beispiel für diese Konvention werden wir in ▶ Abschnitt 4.4.1 begegnen.

4.4.1 Iteration durch Dictionaries

Wie bei Listen, Tupeln und Sets ist eine der häufigsten Aufgaben beim Dictionary die Iteration über seine Elemente. Sie könnten versucht sein, wie folgt über die Schlüssel zu iterieren:

```
>>> for key in moonwalks.keys():    # Nicht pythonisch
...     print(f"{key} unternahm seinen ersten Mondspaziergang ↩
{moonwalks[key]}.")
...
Neil Armstrong unternahm seinen ersten Mondspaziergang 1969.
Buzz Aldrin unternahm seinen ersten Mondspaziergang 1969.
Alan Shepard unternahm seinen ersten Mondspaziergang 1971.
Eugene Cernan unternahm seinen ersten Mondspaziergang 1972.
Michael Jackson unternahm seinen ersten Mondspaziergang 1983.
```

Wie im Kommentar bereits angedeutet, ist dies nicht pythonisch, weil bei der Iteration durch ein Dictionary *standardmäßig* durch die Schlüssel iteriert wird:

```
>>> for key in moonwalks:           # Etwas pythonisch
...     print(f"{key} unternahm seinen ersten Mondspaziergang ↩
{moonwalks[key]}.")
...
Neil Armstrong unternahm seinen ersten Mondspaziergang 1969.
Buzz Aldrin unternahm seinen ersten Mondspaziergang 1969.
Alan Shepard unternahm seinen ersten Mondspaziergang 1971.
Eugene Cernan unternahm seinen ersten Mondspaziergang 1972.
Michael Jackson unternahm seinen ersten Mondspaziergang 1983.
```

Das ist zwar etwas pythonisch, aber wenn Sie sowohl die Schlüssel als auch die Werte verwenden (wie in diesem Fall), ist es sogar besser, mit `items()` über die vollständigen Einträge (die Schlüssel-Wert-Paare) des Dictionary zu iterieren:

```
>>> moonwalks.items()
dict_items([('Neil Armstrong', 1969), ('Buzz Aldrin', 1969), ('Alan
Shepard', 1971), ('Eugene Cernan', 1972), ('Michael Jackson', 1983)])
```

Dies führt zu der in ▶ Listing 4.7 gezeigten eleganten Iteration.

Listing 4.7: Iteration durch die Einträge (`items()`) des Dictionary `moonwalks`.

```
>>> for name, jahr in moonwalks.items():   # Pythonisch
...     print(f"{name} unternahm seinen ersten Mondspaziergang {jahr}.")
...
Neil Armstrong unternahm seinen ersten Mondspaziergang 1969.
Buzz Aldrin unternahm seinen ersten Mondspaziergang 1969.
Alan Shepard unternahm seinen ersten Mondspaziergang 1971.
Eugene Cernan unternahm seinen ersten Mondspaziergang 1972.
Michael Jackson unternahm seinen ersten Mondspaziergang 1983.
```

Beachten Sie, dass wir in ▶ Listing 4.7 mit `name, jahr` anstelle der unspezifischen `key, value` auch zu aussagekräftigen Namen gewechselt haben.

4.4.2 Zusammenführen von Dictionaries

Eine häufige Operation ist das *Zusammenführen* (engl.: *merging*) von Dictionaries, bei der die Elemente zweier Dictionaries zu einem einzigen zusammengefasst werden. Nehmen wir zum Beispiel zwei Dictionaries, die aus Schulfächern mit dazugehörigen Klausurergebnissen bestehen:

```
>>> klausuren1 = {"Mathematik": 75, "Physik": 99}
>>> klausuren2 = {"Geschichte": 77, "Englisch": 93}
```

Es wäre schön, wenn man auf einfache Weise aus diesen beiden Dictionaries ein Dictionary erzeugen könnte, das alle vier Kombinationen von Fächern und Klausurergebnissen enthält.

Ältere Versionen von Python unterstützten das Zusammenführen von Dictionaries nicht, aber in Python 3.5 wurde die **-Syntax für diesen Zweck hinzugefügt:

```
>>> {**klausuren1, **klausuren2}    # Irgendwie pythonisch
{'Mathematik': 75, 'Physik': 99, 'Geschichte': 77, 'Englisch': 93}
```

Wenn Sie mich fragen, ist das eine ziemlich merkwürdige Syntax, die ich hier vor allem deshalb aufführe, weil sie Ihnen im Code anderer Programmierer begegnen könnte. Glücklicherweise gibt es seit Python 3.9 eine großartige Möglichkeit, Dictionaries mithilfe des Pipe-Operators | zusammenzuführen:

```
>>> klausuren1 | klausuren2          # Sehr pythonisch
{'Mathematik': 75, 'Physik': 99, 'Geschichte': 77, 'Englisch': 93}
```

Wenn die Dictionaries keine sich überschneidenden Schlüssel aufweisen, werden beim Zusammenführen einfach alle Schlüssel-Wert-Paare in einem einzigen Dictionary zusammengefasst. Wenn das *zweite* Dictionary jedoch einen oder mehrere Schlüssel mit dem ersten gemeinsam hat, dann haben seine Werte Vorrang. In diesem Fall können wir uns vorstellen, wie auf diese Weise das erste Dictionary mit dem Inhalt des zweiten *aktualisiert* werden kann.[49] Nehmen wir zum Beispiel an, dass wir die Klausurergebnisse aus den Dictionaries `klausuren1` und `klausuren2` mithilfe des Pipe-Operators in einem Dictionary zusammenführen, das wir der Variablen `klausurergebnisse` zuweisen:

```
>>> klausurergebnisse = klausuren1 | klausuren2
{'Mathematik': 75, 'Physik': 99, 'Geschichte': 77, 'Englisch': 93}
```

49 Aus diesem Grund heißt die Methode, mit der Dictionaries (oder besser gesagt Hashes) zusammengeführt werden, in Ruby `update`.

Nehmen wir nun an, dass der Schüler die Klausuren, bei denen er die niedrigsten Punktzahlen erreicht hat, wiederholen darf und diesmal bessere Ergebnisse erzielt:

```
>>> wiederholung = {"Mathematik": 97, "Geschichte": 94}
```

An dieser Stelle können wir die im Dictionary `klausurergebnisse` hinterlegten ursprünglichen Klausurergebnisse mit den aktualisierten Werten aus den Wiederholungsklausuren auf die oben beschriebene Weise aktualisieren (▶ Listing 4.8).

Listing 4.8: Aktualisieren eines Wörterbuchs mit Hilfe einer Zusammenführung.

```
>>> klausurergebnisse | wiederholung
{'Mathematik': 97, 'Physik': 99, 'Geschichte': 94, 'Englisch': 93}
>>> klausurergebnisse
{'Mathematik': 75, 'Physik': 99, 'Geschichte': 77, 'Englisch': 93}
```

Wir sehen, dass die Werte zu den Schlüsseln `"Mathematik"` und `"Geschichte"` mit den Werten aus dem zweiten Dictionary aktualisiert wurden. Beachten Sie aber, dass hier ein neues Dictionary mit den aktualisierten Werten erzeugt wurde, während die ursprünglichen Dictionaries `klausurergebnisse` und `wiederholung` unverändert geblieben sind! Wenn das Dictionary `klausurergebnisse` selbst auch aktualisiert werden soll, muss das Ergebnis der Zusammenführung der Variablen `klausurergebnisse` erneut zugewiesen werden:

```
>>> klausurergebnisse = klausurergebnisse | wiederholung
{'Mathematik': 97, 'Physik': 99, 'Geschichte': 94, 'Englisch': 93}
>>> klausurergebnisse
{'Mathematik': 97, 'Physik': 99, 'Geschichte': 94, 'Englisch': 93}
```

4.4.3 Übungen

1. Definieren Sie ein Dictionary für einen `nutzer` mit den drei Attributen (Schlüsseln) `"nutzername"`, `"passwort"` und `"passwort_bestätigung"`. Wie würden Sie prüfen, ob das Passwort mit der Bestätigung des Passworts übereinstimmt?
2. Wir haben in ▶ Listing 2.29 und ▶ Listing 3.10 gesehen, dass Strings und Listen in Python für Fälle, in denen bei der Iteration jeweils der aktuelle Iterationsindex benötigt wird, die Funktion `enumerate()` bereitstellen. Bestätigen Sie, dass wir die `enumerate()`-Funktion bei der Iteration durch Dictionaries auf analoge Weise nutzen können, wie beispielsweise im Code in ▶ Listing 4.9.
3. Zeigen Sie durch Umkehrung der Reihenfolge der beiden Dictionaries, die mithilfe des Pipe-Operators zusammengeführt werden, in ▶ Listing 4.8, dass das Ergebnis der Zusammenführung von Dictionaries von deren Reihenfolge bei der Verknüpfung mit dem Pipe-Operator abhängt, sodass `d1 | d2` im Allgemeinen nicht dasselbe ist wie `d2 | d1`. Unterscheiden sich die Ergebnisse nur in den Werten der resultierenden Dictionaries? Unter welchen Bedingungen sind die Ergebnisse identisch?

Listing 4.9: Verwendung von `enumerate()` bei der Iteration durch ein Dictionary.

```
>>> for i, (name, jahr) in enumerate(moonwalks.items()):   # Pythonisch
...     print(f"{i+1}. {name} unternahm seinen ersten Mondspaziergang ↵
{jahr}.")
...
1. Neil Armstrong unternahm seinen ersten Mondspaziergang 1969.
2. Buzz Aldrin unternahm seinen ersten Mondspaziergang 1969.
```

```
3. Alan Shepard unternahm seinen ersten Mondspaziergang 1971.
4. Eugene Cernan unternahm seinen ersten Mondspaziergang 1972.
5. Michael Jackson unternahm seinen ersten Mondspaziergang 1983.
```

4.5 Anwendung: Unterschiedliche Wörter zählen

Wenden wir die Dictionaries aus ▶ Abschnitt 4.4 nun auf eine anspruchsvollere Aufgabe an, die schließlich zu unserem bisher längsten Programm führt. Die Aufgabe besteht darin, alle unterschiedlichen Wörter aus einem ziemlich langen Text zu extrahieren und zu zählen, wie oft jedes dieser Wörter in dem Text vorkommt.

Da die Befehlsfolge diesmal etwas umfangreicher ist, arbeiten wir in diesem Fall mit einer Python-Datei (▶ Abschnitt 1.3), die dann mit dem `python3`-Befehl ausgeführt wird. (Wir machen daraus kein eigenständiges Shell-Skript wie in ▶ Abschnitt 1.4, weil wir nicht vorhaben, das Programm als allgemein verwendbares Dienstprogramm zu entwerfen). In jeder Phase empfehle ich Ihnen, den Code interaktiv mit Python auszuführen, wenn Sie Fragen zu den Auswirkungen eines bestimmten Befehls haben.

Beginnen wir mit der Erstellung unserer Datei:

```
(venv) $ touch count.py
```

Füllen Sie die Datei nun mit einem String, der den Text des von uns dafür ausgewählten Sonetts 116[50] (▶ Abbildung 4.9[51]) von Shakespeare enthält, den wir ▶ Listing 4.6 entnommen und in ▶ Listing 4.10 erneut wiedergegeben haben. Stellen Sie außerdem den Import des Moduls `re` voran, da wir im weiteren Verlauf mit regulären Ausdrücken arbeiten müssen.

Listing 4.10: *count.py* Hinzufügen des Imports des Moduls `re` und des zu analysierenden Strings zu unserer Python-Datei.

```
import re


sonnet = """Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove.
O no, it is an ever-fixed mark
That looks on tempests and is never shaken
It is the star to every wand'ring bark,
Whose worth's unknown, although his height be taken.
Love's not time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
   If this be error and upon me proved,
   I never writ, nor no man ever loved."""
```

50 Beachten Sie, dass sich in der ursprünglichen Aussprache, die zu Shakespeares Zeiten verwendet wurde, Wörter wie „love“ und „remove“ reimten, ebenso wie „come“ und „doom“.

51 Bild mit freundlicher Genehmigung von Psychoshadowmaker/123RF.

Abbildung 4.9: Sonnet 116 vergleicht die Beständigkeit der Liebe mit dem Leitstern für eine wandernde Barke (Schiff).

Unser Ziel ist die Erstellung eines Dictionary namens `uniques`, dessen Schlüssel den unterschiedlichen Wörtern im String `sonnet` und dessen Werte der jeweiligen Anzahl der Vorkommen dieser Wörter in `sonnet` entsprechen. Wir beginnen mit der Erstellung des *leeren* Dictionary `uniques`:

```
uniques = {}
```

Für die Zwecke dieser Übung definieren wir ein „Wort“ als eine Folge von einem oder mehreren *Wortzeichen* (d. h. Buchstaben oder Zahlen, aber keine Leerraumzeichen). Die Suche nach solchen Wörtern kann mithilfe eines regulären Ausdrucks (▶ Abschnitt 4.3) erfolgen. Tatsächlich gibt es mit `\w` einen kurzen regulären Ausdruck für genau diesen Fall (▶ Abbildung 4.5):

```
words = re.findall(r"\w+", sonnet)
```

Wir haben hier die Methode `findall()` aus ▶ Abschnitt 4.3 verwendet, um eine Liste aller Teilzeichenketten im String `sonnet` zu erhalten, die mit dem Muster „ein oder mehrere Wortzeichen in einer Reihe“ übereinstimmen. (Die Erweiterung dieses Musters auf Apostrophe, sodass es z. B. auch auf „wand’ring“ passt, verbleibt als Übung für ▶ Abschnitt 4.5.1.)

An diesem Punkt sollte die Datei den in ▶ Listing 4.11 gezeigten Inhalt aufweisen.

Listing 4.11: *count.py* Hinzufügen eines leeren Dictionary und einer Liste mit allen Wörtern des Strings `sonnet`.

```
import re


sonnet = """Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove.
O no, it is an ever-fixed mark
That looks on tempests and is never shaken
It is the star to every wand'ring bark,
Whose worth's unknown, although his height be taken.
Love's not time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
   If this be error and upon me proved,
   I never writ, nor no man ever loved."""

uniques = {}
words = re.findall(r"\w+", sonnet)
```

Nun zum Kernstück unseres Programms. Wir iterieren durch die Liste `words` und tun dabei Folgendes:

1. Wenn das Wort bereits als Schlüssel im Dictionary `uniques` vorkommt, erhöhen wir den zugehörigen Wert, der die Anzahl des Vorkommens des Wortes in `sonnet` repräsentiert, um 1.
2. Wenn das Wort noch nicht als Schlüssel im Dictionary `uniques` enthalten ist, fügen wir das Wort als Schlüssel zu `uniques` hinzu und initialisieren den zugehörigen Wert mit 1.

Das Ergebnis unter Verwendung des Operators +=, den wir in ▶ Abschnitt 4.3 kurz kennengelernt haben, sieht so aus:

```
for word in words:
    if word in uniques:
        uniques[word] += 1
    else:
        uniques[word] = 1
```

Zum Schluss geben wir das Ergebnis auf dem Terminal aus:

```
print(uniques)
```

Das vollständige Programm (mit hinzugefügten Kommentaren) finden Sie in ▶ Listing 4.12.

Listing 4.12: *count.py* Ein Programm zum Auflisten und Zählen von unterschiedlichen Wörtern in einem Text.

```
import re


sonnet = """Let me not to the marriage of true minds
Admit impediments. Love is not love
```

```
Which alters when it alteration finds,
Or bends with the remover to remove.
O no, it is an ever-fixed mark
That looks on tempests and is never shaken
It is the star to every wand'ring bark,
Whose worth's unknown, although his height be taken.
Love's not time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
   If this be error and upon me proved,
   I never writ, nor no man ever loved."""

# Dictionary für unterschiedliche Wörter und ihre Anzahl
uniques = {}
# Alle Wörter in dem Text
words = re.findall(r"\w+", sonnet)

# Iteration durch die Liste words und schrittweises Füllen des ↩
Dictionary uniques
for word in words:
    if word in uniques:
        uniques[word] += 1
    else:
        uniques[word] = 1

print(uniques)
```

Das Ergebnis der Ausführung von `count.py` im Terminal sieht dann in etwa so aus:

```
(venv) $ python3 count.py
{'Let': 1, 'me': 2, 'not': 4, 'to': 4, 'the': 4, 'marriage': 1, 'of': 2,
'true': 1, 'minds': 1, 'Admit': 1, 'impediments': 1, 'Love': 3, 'is': 4,
'love': 1, 'Which': 1, 'alters': 2, 'when': 1, 'it': 3, 'alteration': 1,
'finds': 1, 'Or': 1, 'bends': 1, 'with': 2, 'remover': 1, 'remove': 1,
'O': 1, 'no': 2, 'an': 1, 'ever': 2, 'fixed': 1, 'mark': 1, 'That': 1,
'looks': 1, 'on': 1, 'tempests': 1, 'and': 4, 'never': 2, 'shaken': 1,
'It': 1, 'star': 1, 'every': 1, 'wand': 1, 'ring': 1, 'bark': ↩
1, 'Whose': 1,
'worth': 1, 's': 4, 'unknown': 1, 'although': 1, 'his': 3, 'height': 1,
'be': 2, 'taken': 1, 'time': 1, 'fool': 1, 'though': 1, 'rosy': ↩
1, 'lips': 1,
'cheeks': 1, 'Within': 1, 'bending': 1, 'sickle': 1, 'compass': ↩
1, 'come': 1,
'brief': 1, 'hours': 1, 'weeks': 1, 'But': 1, 'bears': 1, 'out': ↩
1, 'even': 1,
'edge': 1, 'doom': 1, 'If': 1, 'this': 1, 'error': 1, 'upon': ↩
1, 'proved': 1,
'I': 1, 'writ': 1, 'nor': 1, 'man': 1, 'loved': 1}
```

Dies ist ein gutes Beispiel für eine Lösung „von Hand“, die einigermaßen pythonisch ist. Es gibt aber eine noch pythonischere, wenn auch wesentlich fortgeschrittenere Lösung (▶ Abschnitt 4.5.1). Wie in ▶ Exkurs 1.1 erwähnt, ist die Bedeutung der Bezeichnung *pythonisch* fließend und das Programm in ▶ Listing 4.12 ist ein ausgezeichneter Anfang.

4.5.1 Übungen

1. Erweitern Sie den in Listing ▶4.12 verwendeten regulären Ausdruck um einen Apostroph, sodass er z. B. auch auf „wand'ring“ passt. *Tipp:* Verwenden Sie den ersten Ausdruck in der QUICK REFERENCE bei regex101 (▶Abbildung 4.10) mit `\w`, einem Apostroph und dem Plus-Operator +. Testen Sie Ihren regulären Ausdruck auf regex101 (https://regex101.com).

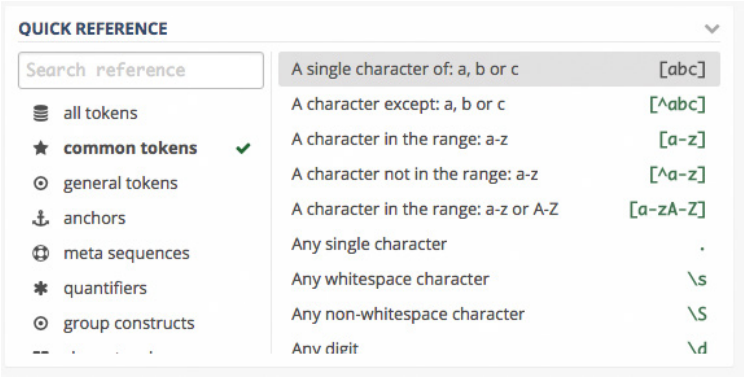

Abbildung 4.10: Ein Tipp für einen regulären Ausdruck.

2. Führen Sie den Code in ▶Listing 4.13 aus und zeigen Sie, dass wir die Ergebnisse aus ▶Listing 4.12 mit der leistungsstarken `Counter()`-Klasse aus dem Python-Modul `collections` auf effektive Weise replizieren können. In diesem hervorragenden Video (https://www.youtube.com/watch?v=8OKTAedgFYg&t=364s) finden Sie weitere Einzelheiten zu diesem Thema.

Listing 4.13: Verwendung der mächtigen `Counter()`-Klasse.

```
import re

from collections import Counter


sonnet = """Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove.
O no, it is an ever-fixed mark
That looks on tempests and is never shaken
It is the star to every wand'ring bark,
Whose worth's unknown, although his height be taken.
Love's not time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
   If this be error and upon me proved,
   I never writ, nor no man ever loved."""

words = re.findall(r"\w+", sonnet)
print(Counter(words))
```

Funktionen und Iteratoren

5

Bislang haben wir in diesem Buch mehrere Beispiele für Python-Funktionen gesehen, die eine der wichtigsten Ideen in Python, ja in der gesamten Informatik sind. In diesem Kapitel lernen wir, wie wir unsere eigenen Funktionen definieren können (▶ Abbildung 5.1). Wir werden auch etwas mehr über Iteratoren lernen (die in ▶ Abschnitt 3.4.2 bereits kurz behandelt wurden), sowohl weil Python solche Objekte oft als Rückgabewerte von eingebauten Funktionen verwendet, als auch weil sie an sich wichtig sind.

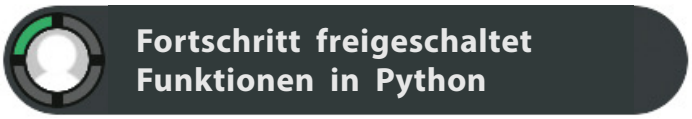

Abbildung 5.1: Es ist Zeit, eine Ebene höher zu steigen.

Falls Sie die Python-Shell nicht bereits ausführen, sollten Sie die virtuelle Umgebung aktivieren und die REPL wie gewohnt starten:

```
$ source venv/bin/activate
(venv) $ python3
```

5.1 Funktionen definieren

Wie wir bereits bei Funktionen wie `print()` (▶ Abschnitt 2.3), `len()` (▶ Abschnitt 2.4) sowie `sorted()` und `reversed()` (▶ Abschnitt 3.4.2) gesehen haben, bestehen Funktionsaufrufe in Python aus einem *Namen* und null oder mehr Argumenten in Klammern:

```
print("hello, world!")
```

Eine der wichtigsten Aufgaben beim Programmieren besteht darin, eigene Funktionen zu definieren, was in Python mit dem Schlüsselwort `def` erfolgen kann. (Wie in ▶ Abschnitt 2.5 beschrieben, werden Funktionen, die an Objekte gebunden sind (wie z. B. `split()` und `islower()`) auch als *Methoden* bezeichnet. Wir werden in ▶ Kapitel 7 lernen, wie wir eigene Methoden definieren können.)

Schauen wir uns ein einfaches Beispiel für die Definition einer Funktion in der REPL an. Wir beginnen mit einer Funktion, die ein einzelnes numerisches Argument annimmt und das Quadrat des Arguments zurückgibt, wie in ▶ Listing 5.1 gezeigt.[52]

Listing 5.1: Definieren einer Funktion.

```
>>> def quadrat(x):
...     return x*x
...
>>> quadrat(10)
100
```

(Hier könnten wir auch `x**2` anstelle von `x*x` verwenden.) Die Funktion endet mit dem Schlüsselwort `return`, gefolgt vom Rückgabewert der Funktion.

52 Python hat keinen Typ-Mechanismus, um z. B. numerische Argumente für Funktionen zu erzwingen. Es gibt eine Bibliothek namens typing mit Unterstützung für Typ-Hinweise.

Copyright