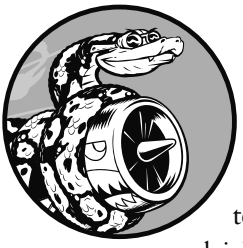
3
Eine Einführung in Listen



In diesem und dem folgenden Kapitel erfahren Sie, was Listen sind und wie Sie mit den darin enthaltenen Elementen arbeiten. In Listen können Sie zusammengehörige Informationen speichern, ganz gleich, ob es nur wenige oder Millionen von Elementen sind. Listen gehören zu den vielseitigsten Merkmalen von Python und können schon von Einsteigern in die Programmierung genutzt werden.

Was sind Listen?

Eine *Liste* ist eine geordnete Sammlung von Elementen. Sie können eine Liste aller Buchstaben des Alphabets, der Ziffern von 0 bis 9 oder aller Mitglieder Ihrer Familie aufstellen. In eine Liste können Sie beliebige Dinge aufnehmen, und die Elemente dieser Liste müssen auch nicht auf eine bestimmte Weise miteinander in Beziehung stehen. Da Listen gewöhnlich mehr als ein Element enthalten, ist es sinnvoll, Listennamen im Plural zu verwenden, also z. B. `letters`, `digits` oder `names`.

In Python werden Listen durch eckige Klammern dargestellt, wobei die einzelnen Elemente in der Liste durch Kommata getrennt sind. Das folgende einfache Beispiel zeigt eine Liste von Fahrradtypen:

bicycles.py

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

Wenn Sie Python anweisen, eine Liste auszugeben, zeigt Python sie in dieser Darstellung einschließlich der eckigen Klammern an:

```
['trek', 'cannondale', 'redline', 'specialized']
```

Das ist allerdings nicht die Ausgabe, die Sie den Benutzern Ihres Programms zumuten wollen. Daher sehen wir uns als Nächstes an, wie Sie auf die einzelnen Elemente in der Liste zugreifen.

Elemente in einer Liste ansprechen

Da es sich bei Listen um geordnete Sammlungen handelt, können Sie auf ein darin enthaltenes Element zugreifen, indem Sie Python dessen Position oder *Index* mitteilen. Dabei geben Sie den Namen der Liste gefolgt vom Index des gewünschten Elements in eckigen Klammern an.

Um das erste Element aus der Liste `bicycles` abzurufen, gehen Sie wie folgt vor:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
❶ print(bicycles[0])
```

Die Syntax für diesen Vorgang sehen Sie bei ❶. Wenn wir nach einem einzelnen Element aus einer Liste fragen, gibt Python ausschließlich dieses Element ohne eckige Klammern oder Anführungszeichen zurück:

```
trek
```

Dies ist ein Ergebnis, wie wir es den Benutzern präsentieren wollen: eine sauber formatierte Ausgabe.

Auf die Elemente in der Liste können Sie auch die Stringmethoden aus Kapitel 2 anwenden. Beispielsweise können wir das Element `'trek'` mit der Methode `title()` noch besser formatieren:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0].title())
```

Dabei erhalten wir das gleiche Ergebnis wie im vorherigen Beispiel, allerdings hat die Ausgabe `Trek` jetzt einen großen Anfangsbuchstaben.

Indizes beginnen bei 0, nicht bei 1

Für Python befindet sich das erste Element einer Liste an der Position 0, nicht 1. Das gilt auch für die meisten anderen Programmiersprachen, was daran liegt, wie Listenoperationen auf einer maschinennahen Ebene verwirklicht werden. Wenn Sie unerwartete Ergebnisse erhalten, sollten Sie als Erstes prüfen, ob Sie bei der Angabe eines Index um eine Stelle daneben liegen.

Das zweite Element in einer Liste hat den Index `1`. Sie können jedes Element in einer Liste abrufen, indem Sie ganz einfach zählen, wo es sich befindet, und dann 1 von diesem Wert abziehen. Um beispielsweise auf das vierte Element einer Liste zuzugreifen, müssen Sie das Element am Index `3` abrufen.

Der folgende Code ruft die Fahrradtypen an den Indizes `1` und `3` ab:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

Zurückgegeben werden das zweite und das vierte Fahrrad:

```
cannondale
specialized
```

In Python gibt es auch eine besondere Schreibweise, um auf das letzte Element einer Liste zuzugreifen. Wenn Sie das Element am Index `-1` abrufen, gibt Python das letzte Element zurück:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

Dieser Code gibt den Wert `specialized` zurück. Das ist eine sehr praktische Schreibweise, denn es kommt oft vor, dass Sie das letzte Element einer Liste brauchen, aber nicht wissen, wie lang sie ist. Dies funktioniert übrigens auch mit anderen negativen Indexwerten. So steht der Index `-2` für das zweitletzte Element der Liste, `-3` für das drittletzte usw.

Einzelne Werte aus einer Liste verwenden

Sie können die einzelnen Werte in einer Liste auch wie jede andere Variable nutzen. Wenn Sie Strings mit F-Strings zu einer Nachricht kombinieren, können Sie darin auch Listenwerte aufnehmen.

Versuchen wir, den ersten Fahrradtyp aus der Liste abzurufen und in eine Nachricht einzubauen:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
❶ message = f"My first bicycle was a {bicycles[0].title()}."

print(message)
```

Bei ❶ bauen wir einen Satz unter Zuhilfenahme des Wertes am Index `bicycles[0]` zusammen und weisen ihn der Variablen `message` zu. Als Ausgabe erhalten wir den folgenden einfachen Satz, in dem das erste Fahrrad in der Liste vorkommt:

```
My first bicycle was a Trek.
```

Probieren Sie es selbst aus!

Versuchen Sie sich an den folgenden kurzen Programmen, um erste Erfahrungen mit Listen in Python zu sammeln. Um den Überblick zu behalten, sollten Sie die Übungen eines Kapitels jeweils in einem eigenen Ordner ablegen.

3-1 Namen: Speichern Sie die Namen einiger Freunde in der Liste `names`. Geben Sie die Namen aller Personen aus, indem Sie nacheinander jedes Element abrufen.

3-2 Grüße: Verwenden Sie wiederum die Liste aus Übung 3-1, aber geben Sie nicht einfach die Namen aus, sondern an die einzelnen Personen gerichtete Nachrichten. Alle diese Nachrichten sollen den gleichen Text aufweisen, aber jeweils eine andere Person ansprechen.

3-3 Ihre eigene Liste: Erstellen Sie eine Liste mit mehreren Beispielen für Ihr bevorzugtes Verkehrsmittel, z. B. Motorräder oder Autos. Geben Sie dann mehrere Aussagen über die Elemente in dieser Liste aus, z. B.: »I would like to own a Honda motorcycle.«

Elemente ändern, hinzufügen und entfernen

Die Listen, die Sie in Ihrem Programm erstellen, sind meistens dynamischer Natur. Das heißt, dass Sie die Liste aufstellen und dann im Programmverlauf Elemente hinzufügen oder daraus entfernen. Stellen Sie sich beispielsweise ein Spiel vor, bei dem man außerirdische Raumschiffe abschießen muss. Sie können die ursprüngliche Menge der Gegner in einer Liste speichern und dann jeweils ein Element entfernen, wenn das entsprechende Schiff abgeschossen wurde. Jedes Mal, wenn ein neues Schiff auftaucht, fügen Sie es der Liste hinzu. Im Verlauf des Spiels wird die Liste daher ständig wachsen und schrumpfen.

Elemente in einer Liste ändern

Für die Änderung von Elementen in einer Liste verwenden Sie eine ähnliche Syntax wie für den Zugriff. Um ein Element zu ändern, geben Sie den Namen der Liste gefolgt von dem Index des Elements an und stellen dann den neuen Wert bereit.

Nehmen wir an, Sie haben eine Liste von Motorrädern, deren erster Wert `'honda'` lautet. Wie können Sie diesen Wert ändern?

motorcycles.py

```
❶ motorcycles = ['honda', 'yamaha', 'suzuki']
  print(motorcycles)

❷ motorcycles[0] = 'ducati'
  print(motorcycles)
```

Bei ❶ wird die ursprüngliche Liste definiert, die `'honda'` als erstes Element enthält. Der Code bei ❷ ändert den Wert dieses ersten Elements in `'ducati'`. Die Ausgabe beweist, dass dieses Element tatsächlich geändert wurde, während der Rest der Liste unverändert geblieben ist:

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

Natürlich können Sie den Wert jedes beliebigen Elements in einer Liste ändern, nicht nur den des ersten.

Elemente zu einer Liste hinzufügen

Es kann viele Gründe dafür geben, eine Liste um weitere Elemente zu ergänzen, etwa um in einem Spiel neue Gegner erscheinen zu lassen, um neue Daten zu einer Darstellung oder neu registrierte Benutzer zu einer Website hinzuzufügen. Python bietet mehrere Möglichkeiten dazu.

Elemente am Ende einer Liste anhängen

Die einfachste Möglichkeit, neue Elemente zu einer Liste hinzuzufügen, besteht darin, sie am Ende *anzuhängen*. Sehen wir uns an, wie wir das Element `'ducati'` hinten an die Liste aus dem vorherigen Beispiel anhängen:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

❶ motorcycles.append('ducati')
  print(motorcycles)
```

Die Methode `append()` bei ❶ fügt `'ducati'` am Ende der Liste hinzu, ohne dass dies Auswirkungen auf die anderen Elemente hätte:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

Die Methode `append()` macht es leicht, Listen dynamisch zu erstellen. Beispielsweise können Sie mit einer leeren Liste beginnen und ihr dann mit einer Folge von `append()`-Anweisungen Elemente hinzufügen wie im folgenden Beispiel:

```
motorcycles = []

motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')

print(motorcycles)
```

Die resultierende Liste sieht genauso aus wie diejenige, die wir in unseren vorherigen Beispielen verwendet haben:

```
['honda', 'yamaha', 'suzuki']
```

Dies ist eine gebräuchliche Vorgehensweise, da Sie oft nicht wissen können, welche Daten Ihre Benutzer in dem Programm speichern werden. Um den Benutzern die volle Kontrolle zu geben, definieren Sie eine leere Liste für deren Werte und hängen Sie dann jeden neuen, von den Benutzern bereitgestellten Wert daran an.

Elemente in eine Liste einfügen

Mit der Methode `insert()` können Sie ein neues Element auch an einer beliebigen Stelle der Liste einfügen. Dazu müssen Sie den gewünschten Index des neuen Elements und dessen Wert angeben:

```
motorcycles = ['honda', 'yamaha', 'suzuki']

❶ motorcycles.insert(0, 'ducati')
  print(motorcycles)
```

Hier fügt der Code bei ❶ den Wert `'ducati'` am Anfang der Liste ein. Die Methode `insert()` räumt am Index `0` Platz frei und speichert dort den neuen Wert. Dadurch werden alle anderen Werte in der Liste um eine Stelle nach rechts verschoben:

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

Elemente aus einer Liste entfernen

Es kommt oft vor, dass Sie ein oder mehrere Elemente aus einer Liste entfernen müssen. Wenn ein Spieler ein gegnerisches Raumschiff abgeschossen hat, müssen Sie es aus der Liste der Schiffe austragen, und wenn ein Benutzer sein Konto für eine Webanwendung aufhebt, muss er ebenfalls aus der Liste der aktiven Benutzer entfernt werden. Elemente können Sie sowohl anhand ihres Index als auch anhand ihres Wertes entfernen.

Elemente mit der Anweisung del entfernen

Wenn Sie die Position des Elements kennen, das Sie entfernen möchten, können Sie die Anweisung `del` verwenden:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

❶ del motorcycles[0]
print(motorcycles)
```

Der Code bei ❶ entfernt mithilfe von `del` das erste Element aus der Liste der Motorräder, nämlich `'honda'`:

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

Mit `del` können Sie Elemente an beliebigen Stellen in einer Liste entfernen, solange sie nur deren Index kennen. Im folgenden Beispiel nutzen wir diese Anweisung, um das zweite Element zu löschen, `'yamaha'`:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

del motorcycles[1]
print(motorcycles)
```

Tatsächlich wird das zweite Motorrad aus der Liste ausgetragen:

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

In beiden Fällen ist es anschließend nicht mehr möglich, auf den entfernten Wert zuzugreifen.

Elemente mit der Methode pop() entfernen

Manchmal möchten Sie den Wert eines Elements noch verwenden, nachdem Sie es aus der Liste entfernt haben, beispielsweise um die x- und y-Koordinaten eines abgeschossenen Raumschiffs zu ermitteln und eine Explosion an der betreffenden Stelle zu zeichnen. Bei einer Webanwendung kann es sein, dass Sie einen Benutzer aus der Liste der aktiven Mitglieder herausnehmen, aber zur Liste der inaktiven Mitglieder hinzufügen möchten.

Die Methode `pop()` entfernt zwar den letzten Eintrag aus einer Liste, lässt es aber zu, weiterhin damit zu arbeiten. Probieren wir das mit einem Motorrad in unserer Liste aus:

```
❶ motorcycles = ['honda', 'yamaha', 'suzuki']
  print(motorcycles)

❷ popped_motorcycle = motorcycles.pop()
❸ print(motorcycles)
❹ print(popped_motorcycle)
```

Bei ❶ erstellen wir die Liste `motorcycles` und geben Sie aus. Bei ❷ entfernen wir den letzten Wert mit `pop()` und speichern diesen Wert in der Variablen `popped_motorcycle`. Anschließend geben wir bei ❸ die Liste aus, um zu zeigen, dass tatsächlich ein Wert entfernt wurde. Diesen Wert geben wir nun bei ❹ aus, was beweist, dass wir ihn nach wie vor verwenden können.

Die Ausgabe zeigt, dass der Wert `'suzuki'` vom Ende der Liste entfernt wurde und sich jetzt in der Variablen `popped_motorcycle` befindet:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

Welchen Nutzen hat diese Methode in der Praxis? Nehmen wir an, die Motorräder in der Liste sind in chronologischer Reihenfolge nach dem Kaufdatum geordnet. Dann können wir mithilfe von `pop()` eine Aussage über das zuletzt gekaufte Motorrad ausgeben:

```
motorcycles = ['honda', 'yamaha', 'suzuki']

last_owned = motorcycles.pop()
print("The last motorcycle I owned was a " + last_owned.title() + ".")
```

Als Ausgabe erhalten wir einen Satz über das letzte Motorrad, das ich besessen habe:

```
The last motorcycle I owned was a Suzuki.
```

Elemente mit pop() von beliebigen Stellen einer Liste entfernen

Mit `pop()` können Sie auch ein Element an jeder Stelle in der Liste entfernen, indem Sie in den Klammern den Index dieses Elements angeben.

```
motorcycles = ['honda', 'yamaha', 'suzuki']

❶ first_owned = motorcycles.pop(0)
❷ print(f"The first motorcycle I owned was a {first_owned.title()}.")
```

Hier nehmen wir bei ❶ das erste Motorrad aus der Liste heraus und geben dann bei ❷ eine Nachricht über dieses Motorrad aus. Als Ausgabe erhalten wir einen Satz über das erste Motorrad, das ich besessen habe:

```
The first motorcycle I owned was a Honda.
```

Denken Sie bei der Verwendung von `pop()` daran, dass das Element, mit dem Sie arbeiten, nicht mehr in der Liste gespeichert ist.

Wann sollten Sie die Anweisung `del` verwenden und wann die Methode `pop()`? Wenn Sie ein Element aus einer Liste entfernen möchten und es danach nicht mehr verwenden wollen, nehmen Sie `del`; wollen Sie jedoch das entfernte Element weiterhin nutzen, entscheiden Sie sich für `pop()`.

Elemente anhand ihres Wertes entfernen

Es kann vorkommen, dass Sie nicht wissen, an welcher Position das Element steht, das Sie entfernen möchten. Wenn Sie aber seinen Wert kennen, können Sie die Methode `remove()` verwenden.

Nehmen wir an, Sie wollen den Wert `'ducati'` aus der Liste der Motorräder entfernen:

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)

❶ motorcycles.remove('ducati')
print(motorcycles)
```

Der Code bei ❶ weist Python an, herauszufinden, wo `'ducati'` in der Liste vorkommt, und das betreffende Element dann zu entfernen:

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

Bei der Verwendung von `remove()` gibt es eine Möglichkeit, um mit dem Wert weiterzuarbeiten, den Sie aus der Liste entfernt haben. Im folgenden Beispiel wollen wir abermals `'ducati'` aus der Liste herausnehmen, aber auch einen Grund dafür angeben:

```
❶ motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
  print(motorcycles)

❷ too_expensive = 'ducati'
❸ motorcycles.remove(too_expensive)
  print(motorcycles)
❹ print(f"\nA {too_expensive.title()} is too expensive for me.")
```

Nach der Definition der Liste (❶) weisen wir den Wert `'ducati'` der Variablen `too_expensive` zu (❷). Anschließend verwenden wir diese Variable, um Python bei ❸ anzuweisen, welchen Wert es von der Liste entfernen soll. Bei ❹ steht der Wert `'ducati'` schon nicht mehr in der Liste, ist aber immer noch über die Variable `too_expensive` erreichbar, sodass wir eine Nachricht darüber ausgeben können, warum wir diesen Wert aus der Liste herausgenommen haben:

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']

A Ducati is too expensive for me.
```

Hinweis

Die Methode `remove()` entfernt nur das erste Vorkommen des angegebenen Wertes. Wenn die Möglichkeit besteht, dass dieser Wert mehrmals in der Liste vorkommt, müssen Sie eine Schleife verwenden, um festzustellen, ob alle Vorkommen entfernt wurden. Wie Sie das tun, erfahren Sie in Kapitel 7.

Probieren Sie es selbst aus!

Die folgenden Übungen sind ein wenig anspruchsvoller als diejenigen aus Kapitel 2, aber sie geben Ihnen die Gelegenheit, Listen auf alle zuvor beschriebenen Weisen anzuwenden.

3-4 Gästeliste: Erstellen Sie eine Liste von mindestens drei Personen (ob lebend oder bereits verstorben), die Sie gern zum Abendessen einladen möchten. Geben Sie dann für jede dieser Personen eine Nachricht mit der Einladung aus.

→

3-5 Gästeliste ändern: Sie haben gerade erfahren, dass einer der Gäste nicht zum Abendessen kommen kann, weshalb Sie eine weitere Person einladen und einen neuen Satz Einladungen verschicken müssen.

- Beginnen Sie mit dem Programm aus Übung 3-4. Fügen Sie am Ende eine `print`-Anweisung mit dem Namen des Gastes hinzu, der nicht kommen kann.
- Ersetzen Sie in der Liste den Namen des Gastes, der nicht kommen kann, durch den der neu eingeladenen Person.
- Geben Sie einen zweiten Satz Einladungen für jede der Personen aus, die noch in der Liste aufgeführt werden.

3-6 Weitere Gäste: Sie haben gerade einen größeren Esstisch entdeckt, sodass Sie jetzt mehr Platz haben. Laden Sie noch drei weitere Gäste zum Abendessen ein.

- Beginnen Sie mit dem Programm aus Übung 3-4 oder 3-5. Fügen Sie am Ende eine `print`-Anweisung hinzu, mit der Sie die Gäste darüber informieren, dass Sie einen größeren Esstisch gefunden haben.
- Fügen Sie mit `insert()` einen neuen Gast am Anfang der Liste ein.
- Fügen Sie mit `insert()` einen neuen Gast in der Mitte der Liste ein.
- Fügen Sie mit `append()` einen Gast am Ende der Liste hinzu.
- Geben Sie einen neuen Satz Einladungen für jede der Personen auf der Liste aus.

3-7 Die Gästeliste verkleinern: Gerade haben Sie erfahren, dass Ihr neuer Esstisch nicht mehr rechtzeitig geliefert werden kann. Sie haben nur Platz für zwei Gäste.

- Beginnen Sie mit dem Programm aus Übung 3-6. Fügen Sie am Ende eine `print`-Anweisung hinzu, mit der Sie die Gäste darüber informieren, dass Sie nur noch zwei Personen einladen können.
- Entfernen Sie mit `pop()` nach und nach Gäste von der Liste, bis nur noch zwei Namen übrig sind. Geben Sie bei jeder Verwendung von `pop()` eine Nachricht an die betreffende Person aus, um sich dafür zu entschuldigen, dass Sie sie nicht zum Essen einladen können.
- Geben Sie an die beiden verbliebenen Personen auf der Liste jeweils eine Nachricht aus, um ihnen mitzuteilen, dass sie nach wie vor eingeladen sind.
- Löschen Sie mit `del` die beiden Namen von der Liste, sodass diese jetzt leer ist. Geben Sie die Liste aus, um sich zu vergewissern, dass Sie am Ende des Programms tatsächlich leer ist.

Listen ordnen

Da Sie keine Kontrolle darüber haben, in welcher Reihenfolge die Benutzer Ihres Programms die Daten bereitstellen, sind Listen oft willkürlich geordnet. Das lässt sich meistens nicht vermeiden, und manchmal kann es auch sinnvoll sein, die ursprüngliche Reihenfolge der Liste beizubehalten. Es gibt jedoch auch Fälle, in denen die Informationen in einer bestimmten Reihenfolge vorliegen müssen. Daher bietet Python verschiedene Möglichkeiten, um Listen zu ordnen.

Listen mit sort() dauerhaft sortieren

Mit der Python-Methode `sort()` ist es recht einfach, Listen zu sortieren. Stellen Sie sich vor, Sie wollen eine Liste von Autos alphabetisch ordnen. Der Einfachheit halber wollen wir annehmen, dass alle Werte in der Liste kleingeschrieben sind.

cars.py

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()                                          ❶
print(cars)
```

Die Methode `sort()` bei ❶ ändert die Reihenfolge der Liste dauerhaft. Die Autos sind jetzt alphabetisch angeordnet, und es ist nicht mehr möglich, zur ursprünglichen Sortierung zurückzukehren:

```
['audi', 'bmw', 'subaru', 'toyota']
```

Um eine Liste in umgekehrter alphabetischer Reihenfolge zu ordnen, übergeben Sie der Methode `sort()` das Argument `reverse=True`, wie das folgende Beispiel zeigt:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

Auch dabei wird die Reihenfolge der Liste dauerhaft geändert:

```
['toyota', 'subaru', 'bmw', 'audi']
```

Listen mit der Funktion sorted() vorübergehend sortieren

Um eine Liste sortiert auszugeben, aber intern die ursprüngliche Reihenfolge beizubehalten, können Sie die Funktion `sorted()` verwenden. Das wollen wir an unserer Autoliste ausprobieren:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']

❶ print("Here is the original list:")
print(cars)

❷ print("\nHere is the sorted list:")
print(sorted(cars))

❸ print("\nHere is the original list again:")
print(cars)
```

Bei ❶ geben wir die Liste in der ursprünglichen Reihenfolge aus, bei ❷ alphabetisch geordnet. Anschließend zeigen wir bei ❸, dass die Originalsortierung der Liste erhalten geblieben ist.

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']

Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']

❹ Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

Wie Sie bei ❹ sehen, existiert die Liste auch nach der Anwendung der Funktion `sorted()` in der ursprünglichen Reihenfolge. Auch `sorted()` akzeptiert übrigens das Argument `reverse=True`, um die Liste in umgekehrter alphabetischer Reihenfolge auszugeben.

Hinweis

Die alphabetische Sortierung einer Liste ist komplizierter, wenn die Werte nicht alle in Kleinbuchstaben vorliegen. Es gibt verschiedene Möglichkeiten für den Umgang mit Großbuchstaben. Wie Sie die gewünschte Reihenfolge genau festlegen, ist etwas aufwendiger, weshalb wir uns hier noch nicht damit beschäftigen. Die meisten Sortiermöglichkeiten bauen jedoch auf dem auf, was Sie in diesem Abschnitt gelernt haben.

Listen in umgekehrter Reihenfolge ausgeben

Mit der Methode `reverse()` können Sie die ursprüngliche Liste in umgekehrter Reihenfolge ausgeben. Wenn wir beispielsweise die Liste der Autos ursprünglich chronologisch nach Anschaffungsdatum geordnet haben, können wir sie damit ganz einfach vom neuesten zum ältesten Auto sortieren:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)

cars.reverse()
print(cars)
```

Beachten Sie, dass `reverse()` keine umgekehrte alphabetische Sortierung vornimmt, sondern einfach die ursprüngliche Reihenfolge der Liste umkehrt:

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

Die Reihenfolge der Liste wird dabei zwar dauerhaft geändert, allerdings können Sie jederzeit zur ursprünglichen Sortierung zurückkehren, indem Sie `reverse()` ein weiteres Mal anwenden.

Die Länge einer Liste ermitteln

Die Länge einer Liste können Sie ganz schnell mit der Funktion `len()` herausfinden. Da die Liste in unserem Beispiel vier Elemente enthält, beträgt ihre Länge 4:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> len(cars)
4
```

Diese Funktion ist sehr praktisch, wenn Sie beispielsweise die Anzahl der Gegner bestimmen müssen, die in einem Spiel noch abgeschossen werden müssen, die Menge der Daten, die noch zu visualisieren sind, oder die Anzahl der registrierten Benutzer einer Website, um nur einige mögliche Aufgaben zu nennen.

Hinweis

Beim Zählen der Elemente in einer Liste beginnt Python bei 1, sodass es bei der Längenbestimmung nicht zu Versatzfehlern kommen kann.

Probieren Sie es selbst!

3-8 Weltreise: Stellen Sie sich fünf Orte vor, die Sie gern besuchen möchten.

- Speichern Sie die Orte in einer Liste, und zwar absichtlich nicht in alphabetischer Reihenfolge.
- Geben Sie die Liste in der ursprünglichen Reihenfolge aus. Machen Sie sich hier keine Gedanken über die Formatierung, sondern geben Sie einfach die rohe Python-Liste aus.

→

- Geben Sie die Liste mithilfe von `sorted()` in alphabetischer Reihenfolge aus, ohne die eigentliche Liste zu verändern.
- Beweisen Sie, dass die Liste immer noch in ihrer ursprünglichen Reihenfolge vorliegt, indem Sie sie ausgeben.
- Geben Sie die Liste mithilfe von `sorted()` in umgekehrter alphabetischer Reihenfolge aus, ohne die eigentliche Liste zu verändern.
- Beweisen Sie, dass die Liste immer noch in ihrer ursprünglichen Reihenfolge vorliegt, indem Sie sie erneut ausgeben.
- Kehren Sie die Reihenfolge der Liste mit `reverse()` um. Geben Sie die Liste aus, um zu beweisen, dass sich die Sortierung geändert hat.
- Kehren Sie die Reihenfolge der Liste erneut mit `reverse()` um. Geben Sie die Liste aus, um zu beweisen, dass sie wieder die ursprüngliche Sortierung hat.
- Sortieren Sie die Liste mit `sort()` alphabetisch. Geben Sie die Liste aus, um zu beweisen, dass sich die Reihenfolge geändert hat.
- Sortieren Sie die Liste mit `sort()` umgekehrt alphabetisch. Geben Sie die Liste aus, um zu beweisen, dass sich die Reihenfolge geändert hat.

3-9 Gäste zum Abendessen: Verwenden Sie in den Programmen aus den Übungen 3-4 bis 3-7 die Funktion `len()`, um eine Nachricht mit der Anzahl der geladenen Gäste auszugeben.

3-10 Alle Funktionen: Überlegen Sie sich ein Thema, zu dem Sie eine Liste erstellen können, beispielsweise eine Liste von Bergen, Flüssen, Ländern, Städten, Sprachen usw. Schreiben Sie ein Programm, das eine Liste solcher Elemente zusammenstellt, und wenden Sie dann jede der in diesem Kapitel besprochenen Funktionen mindestens einmal an.

Indexfehler vermeiden

Wenn Sie beginnen, mit Listen zu arbeiten, wird ein ganz bestimmter Fehler häufig auftreten. Nehmen wir an, Sie haben eine Liste mit drei Elementen und sind an dem dritten interessiert. Mit dem folgenden Code versuchen Sie aber in Wirklichkeit, ein viertes Element abzurufen:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[3])
```

Dies hat einen *Indexfehler* zur Folge:

```
Traceback (most recent call last):
  File "motorcycles.py", line 2, in <module>
```

```
    print(motorcycles[3])
IndexError: list index out of range
```

Python versucht, das Element am Index 3 zurückzugeben, kann dort aber keines finden. Da die Nummerierung der Indizes in Listen um eine Stelle versetzt ist, kommt dieser Fehler häufig vor. Man glaubt, dass das dritte Element den Index 3 hat, doch da Python bei der Indizierung bei 0 beginnt, hat es in Wirklichkeit den Index 2.

Wenn ein solcher Indexfehler in einem Ihrer Programme vorkommt, müssen Sie den abgefragten Indexwert um 1 verringern. Führen Sie das Programm erneut aus, um zu prüfen, ob es jetzt korrekt funktioniert.

Um auf das letzte Element einer Liste zuzugreifen, geben Sie den Index -1 an. Das funktioniert immer, auch wenn sich die Länge der Liste zwischendurch geändert hat:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[-1])
```

Der Index -1 gibt immer das letzte Element in einer Liste zurück, in diesem Fall also 'suzuki':

```
'suzuki'
```

Ein Fehler kann nur in einem einzigen Fall auftreten, nämlich dann, wenn Sie versuchen, das letzte Element einer leeren Liste abzurufen:

```
motorcycles = []
print(motorcycles[-1])
```

Da motorcycles keinerlei Elemente enthält, gibt Python einen weiteren Indexfehler zurück:

```
Traceback (most recent call last):
  File "motorcycles.py", line 3, in <module>
    print(motorcycles[-1])
IndexError: list index out of range
```

Hinweis

Wenn ein Indexfehler auftritt und Sie nicht herausfinden können, wie Sie ihn lösen sollen, geben Sie die Liste oder einfach nur deren Länge aus. Möglicherweise sieht die Liste ganz anders aus, als Sie gedacht haben, insbesondere dann, wenn sie im Programmverlauf dynamisch verändert wird. Die tatsächliche Liste vor Augen zu haben oder wenigstens die genaue Anzahl ihrer Elemente zu kennen, kann helfen, den Fehler zu finden.

Probieren Sie es selbst aus!

3-11 Absichtlicher Fehler: Wenn Ihnen in Ihren Programmen noch kein Indexfehler unterlaufen ist, versuchen Sie, einen solchen zu provozieren. Ändern Sie einen Index in einem Ihrer Programme, um einen Indexfehler hervorzurufen. Beheben Sie den Fehler wieder, bevor Sie das Programm schließen.

Zusammenfassung

In diesem Kapitel haben Sie erfahren, was Listen sind und wie Sie mit den einzelnen Elementen in einer Liste arbeiten können. Sie haben gelernt, wie Sie eine Liste definieren, wie Sie Elemente hinzufügen und entfernen, wie Sie Listen vorübergehend für die Ausgabe, aber auch dauerhaft umsortieren, wie Sie die Länge einer Liste herausfinden und wie Sie Indexfehler vermeiden können.

In Kapitel 4 erfahren Sie, wie Sie mit nur wenigen Zeilen Code die Elemente einer Liste in einer Schleife durchlaufen können. Das ermöglicht Ihnen, effizient zu arbeiten – selbst bei Listen, die Tausende oder gar Millionen von Elementen enthalten.