7

Benutzereingaben und while-Schleifen

Die meisten Programme dienen dazu, ein Problem eines Benutzers zu lösen. Um dazu in der Lage zu sein, muss das Programm aber gewöhnlich zunächst einige Informationen vom Benutzer einholen. Nehmen wir an, jemand möchte herausfinden, ob er alt genug ist, um zu wählen. Wenn Sie ein Programm zur Beantwortung dieser Frage schreiben, müssen Sie das Alter des Benutzers kennen, bevor Sie die Frage beantworten können. Das Programm muss den Benutzer daher auffordern, sein Alter einzugeben. Daraufhin kann es diese Eingabe mit dem Mindestalter zum Wählen vergleichen und dadurch bestimmen, ob der Benutzer alt genug ist, und das Resultat ausgeben.

In diesem Kapitel lernen Sie, wie Sie Benutzereingaben entgegennehmen, um sie in Ihren Programmen zu verarbeiten. Wenn das Programm den Namen des Benutzers benötigt, fordern Sie ihn auf, seinen Namen anzugeben. Braucht das Programm eine Liste von Namen, fordern Sie den Benutzer auf, mehrere Namen einzugeben. Für all das verwenden Sie die Funktion `input()`.

Des Weiteren erfahren Sie, wie Sie Programme so lange laufen lassen, wie die Benutzer damit arbeiten wollen. Mit einer `while`-Schleife können Sie dafür sorgen, dass ein Programm etwas macht, solange eine bestimmte Bedingung erfüllt bleibt. Mit diesen Möglichkeiten können Sie interaktive Programme schreiben.

Die Funktion input()

Die Funktion `input()` hält das Programm an und wartet darauf, dass der Benutzer Text eingibt. Die Eingabe speichert Python dann in einer Variablen, sodass Sie anschließend damit arbeiten können.

Das folgende Beispielprogramm bittet den Benutzer, Text einzugeben, und zeigt dem Benutzer dann diesen Text an:

parrot.py

```
message = input("Tell me something, and I will repeat it back to you: ")
print(message)
```

Die Funktion `input()` besitzt nur ein Argument, und zwar eine *Eingabeaufforderung*, also die Anweisung, die Sie anzeigen lassen wollen, damit die Benutzer wissen, was sie tun sollen. In diesem Beispiel sieht der Benutzer die Eingabeaufforderung *Tell me something, and I will repeat it back to you:*, wenn Python die erste Zeile ausführt. Das Programm wartet dann, sodass der Benutzer etwas eingeben kann, und fährt fort, wenn die Eingabetaste gedrückt wurde. Die Benutzereingabe wird in der Variablen `message` gespeichert und dann mit `print(message)` wieder ausgegeben:

```
Tell me something, and I will repeat it back to you: Hello everyone!
Hello everyone!
```

Hinweis

Sublime Text kann Programme, die Benutzereingaben entgegennehmen, nicht ausführen. Es ist möglich, solche Programme in diesem Editor zu schreiben, aber um sie auszuführen, müssen Sie das Terminal verwenden (siehe »Python-Programme im Terminal ausführen« in Kapitel 1).

Klar verständliche Eingabeaufforderungen schreiben

Wenn Sie die Funktion `input()` verwenden, müssen Sie jeweils eine deutliche, leicht verständliche Eingabeaufforderung schreiben, die den Benutzern genau mitteilt, welche Art von Information Sie benötigen. Betrachten Sie dazu das folgende Beispiel:

greeter.py

```
name = input("Please enter your name: ")
print("Hello, " + name + "!")
```

Geben Sie am Ende der Eingabeaufforderung ein Leerzeichen ein (in diesem Beispiel hinter dem Doppelpunkt), um die Eingabeaufforderung von der Benutzereingabe abzusetzen und deutlich zu machen, wo die Eingabe erfolgen soll:

```
Please enter your name: Eric
Hello, Eric!
```

Es kann vorkommen, dass Sie eine Eingabeaufforderung schreiben, die länger als eine Zeile ist, etwa um dem Benutzer mitzuteilen, warum Sie eine bestimmte Eingabe benötigen. In einem solchen Fall können Sie die Eingabeaufforderung in einer Variablen speichern und diese Variable an die Funktion `input()` übergeben. Dadurch können Sie die Eingabeaufforderung aus mehreren Zeilen aufbauen, die `input()`-Anweisung aber trotzdem übersichtlich halten.

greeter.py

```
prompt = "If you tell us who you are, we can personalize the messages you see."
prompt += "\nWhat is your first name? "

name = input(prompt)
print("\nHello, " + name + "!")
```

Dieses Beispiel zeigt eine Möglichkeit, um einen String aus mehreren Zeilen aufzubauen. In der ersten Zeile wird der erste Teil der Nachricht in der Variablen `prompt` gespeichert, und in der zweiten wird mithilfe des Operators `+=` der neue String ans Ende des Inhalts dieser Variablen angehängt.

Die Eingabeaufforderung umfasst jetzt zwei Zeilen. Auch hier steht der Übersichtlichkeit halber wieder ein Leerzeichen hinter dem Fragezeichen:

```
If you tell us who you are, we can personalize the messages you see.
What is your first name? Eric

Hello, Eric!
```

Verwendung von int() für numerische Eingaben

Bei der Verwendung der Funktion `input()` interpretiert Python alles, was die Benutzer eingeben, als String. Schauen Sie sich die folgende Interpretersitzung an, in der wir uns nach dem Alter des Benutzers erkundigen:

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age
'21'
```

Der Benutzer gibt die Zahl 21 ein, doch wenn wir Python nach dem Wert von `age` fragen, gibt es `'21'` zurück – die Stringdarstellung dieses Wertes, wie wir an den Anführungszeichen erkennen können. Wenn Sie einfach nur die Eingabe anzeigen lassen wollen, ist das kein Problem, aber wenn Sie sie als Zahl weiterverarbeiten, tritt ein Fehler auf:

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age >= 18
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: str() >= int()
```

Die Verwendung der Eingabe in einem numerischen Vergleich (❶) führt zu einem Fehler, da Python einen String wie `'21'` nicht mit einem Integer wie `18` vergleichen kann (❷).

Dieses Problem können wir mithilfe der Funktion `int()` umgehen, die Python anweist, die Stringdarstellung einer Zahl in den numerischen Wert umzuwandeln:

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age = int(age)
>>> age >= 18
True
```

In diesem Beispiel geben wir wieder `21` ein, was Python auch diesmal als String interpretiert. Dann aber wandeln wir diesen String bei ❶ mithilfe von `int()` in einen numerischen Wert um. Nun kann Python prüfen, ob die Variable `age` (in der jetzt der numerische Wert `21` gespeichert ist) größer oder gleich `18` ist, was in diesem Fall `True` ergibt.

Wie können Sie die Funktion `int()` in einem Programm in der Praxis einsetzen? Stellen Sie sich ein Programm vor, das ermittelt, ob jemand groß genug ist, um mit der Achterbahn zu fahren:

rollercoaster.py

```
height = input("How tall are you, in inches? ")
height = int(height)

if height >= 36:
    print("\nYou're tall enough to ride!")
else:
    print("\nYou'll be able to ride when you're a little older.")
```

Das Programm kann `height` mit `36` vergleichen, da die Eingabe durch `height = int(height)` vor dem Vergleich in einen numerischen Wert umgewandelt wurde.

Ist die eingegebene Zahl größer oder gleich 36, teilen wir dem Benutzer mit, dass er groß genug ist:

```
How tall are you, in inches? 71

You're tall enough to ride!
```

Wenn Sie Zahleneingaben in Berechnungen und Vergleichen verwenden, müssen Sie sie zuvor in numerische Werte umwandeln.

Der Modulo-Operator

Ein nützliches Werkzeug für den Umgang mit numerischen Daten ist der *Modulo-Operator* (`%`), der eine Zahl durch eine andere teilt und den Rest zurückgibt:

```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> 7 % 3
1
```

Der Modulo-Operator sagt Ihnen nicht, wie oft eine Zahl in die andere passt, sondern nennt nur den Rest der Division.

Ist eine Zahl durch eine andere teilbar, ist der Rest 0. Damit können Sie herausfinden, ob eine Zahl gerade oder ungerade ist:

even_or_odd.py

```
number = input("Enter a number, and I'll tell you
                if it's even or odd: ")
number = int(number)

if number % 2 == 0:
    print("\nThe number " + str(number) + " is even.")
else:
    print("\nThe number " + str(number) + " is odd.")
```

Gerade Zahlen sind durch 2 teilbar. Ist also der Modulo von einer Zahl und 2 gleich 0 (`if number % 2 == 0`), handelt es sich um eine gerade Zahl, anderenfalls um eine ungerade.

```
Enter a number, and I'll tell you if it's even or odd:

The number 42 is even.
```

Benutzereingaben in Python 2.7

In Python 2.7 müssen Sie mit der Funktion `raw_input()` nach Benutzereingaben fragen. Ebenso wie `input()` in Python 3 interpretiert auch `raw_input()` alle Eingaben als Strings.

Es gibt in Python 2.7 auch eine Funktion namens `input()`, die aber Eingaben als Python-Code interpretiert und auszuführen versucht. Bestenfalls gibt Python eine Fehlermeldung aus, wenn es die Eingabe nicht versteht. Es kann jedoch auch vorkommen, dass Code ausgeführt wird, den Sie gar nicht ausführen lassen wollen. Wenn Sie Python 2.7 verwenden, müssen Sie daher `raw_input()` statt `input()` verwenden.

Probieren Sie es selbst aus!

7-1 Leihwagen: Schreiben Sie ein Programm, das die Benutzer fragt, welche Art von Leihwagen sie gern hätten, und dann eine Nachricht über dieses Auto ausgibt, z. B. etwas wie *Let me see if I can find you a Subaru*.

7-2 Restaurantplätze: Schreiben Sie ein Programm, das die Benutzer fragt, mit wie vielen Personen sie ein Restaurant besuchen möchten. Ist die angegebene Zahl größer als 8, geben Sie die Meldung aus, dass die Gruppe auf einen Tisch warten muss. Anderenfalls teilen Sie mit, dass der Tisch bereitsteht.

7-3 Vielfache von 10: Fragen Sie den Benutzer nach einer Zahl und teilen Sie ihm dann mit, ob es sich dabei um ein Vielfaches von 10 handelt oder nicht.

while-Schleifen

Die `for`-Schleife nimmt eine Zusammenstellung von Elementen entgegen und führt einen Codeblock einmal für jedes der darin enthaltenen Elemente aus. Dagegen läuft eine `while`-Schleife so lange, wie eine bestimmte Bedingung wahr ist.

while-Schleifen in Aktion

Mit einer `while`-Schleife können Sie eine Folge von Zahlen abzählen. Beispielsweise zählt die folgende Schleife von 1 bis 5:

counting.py

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

In der ersten Zeile legen wir den Anfangswert fest, indem wir `current_number` auf 1 setzen. Die `while`-Schleife richten wir so ein, dass sie so lange läuft, wie der Wert von `current_number` kleiner oder gleich 5 ist. Der Code innerhalb der Schleife gibt den Wert von `current_number` aus und setzt den Wert dieser Variablen mit `current_number += 1` um 1 herauf. (Der Operator `+=` ist eine Kurzschreibweise für `current_number = current_number + 1`.)

Python wiederholt die Schleife, solange die Bedingung `current_number <= 5` wahr ist. Da 1 kleiner als 5 ist, gibt Python 1 aus und setzt den Variablenwert um 1 herauf, sodass sich 2 ergibt. Auch 2 ist kleiner als 5, weshalb Python 2 ausgibt und den Wert erneut hochsetzt usw. Sobald der Wert von `current_number` 5 überschreitet, wird die Schleife nicht mehr ausgeführt und das Programm beendet.

```
1
2
3
4
5
```

Die Programme, die Sie täglich verwenden, nutzen sehr wahrscheinlich `while`-Schleifen. Beispielsweise wird ein Spiel mithilfe einer `while`-Schleife so lange ausgeführt, bis Sie den Befehl zum Beenden geben. Es wäre ziemlich frustrierend, wenn Programme einfach aufhörten, ohne dass wir die Anweisung dazu erteilt hätten, oder einfach weiterliefen, auch wenn wir sie beenden wollen. Daher sind `while`-Schleifen sehr praktisch.

Programmbeendigung durch den Benutzer

Wir können dafür sorgen, dass ein Programm so lange ausgeführt wird, wie der Benutzer es wünscht, indem wir den Großteil des Codes in eine `while`-Schleife stellen. Dabei definieren wir einen *Beendigungswert* und lassen das Programm laufen, solange der Benutzer diesen Wert nicht eingibt:

parrot.py

```
❶ prompt = "\nTell me something, and I will repeat it back to you:"
  prompt += "\nEnter 'quit' to end the program. "
❷ message = ""
❸ while message != 'quit':
      message = input(prompt)
      print(message)
```

Bei ❶ definieren wir eine Eingabeaufforderung, mit der wir den Benutzern die beiden möglichen Vorgehensweisen erklären: Sie können entweder einen zu wiederholenden Satz oder den Beendigungswert (hier: `'quit'`) eingeben. Anschließend

richten wir bei ❷ die Variable `message` ein, in der wir den vom Benutzer eingegebenen Wert speichern. Dabei definieren wir `message` als `""`, also als leeren String. Warum ist das notwendig? Wenn Python zum ersten Mal die `while`-Aussage erreicht, muss es den Wert von `message` mit `'quit'` vergleichen, aber zu diesem Zeitpunkt liegt noch keine Benutzereingabe vor. Da das Programm nicht weiterlaufen kann, wenn Python nichts zu vergleichen hat, geben wir `message` einen Anfangswert. Es ist zwar nur ein leerer String, aber das reicht für Python aus, um den Vergleich vorzunehmen und die `while`-Schleife auszuführen. Diese Schleife läuft, solange der Wert von `message` nicht `'quit'` ist (❸).

Da `message` beim ersten Durchlauf nur einen leeren String enthält, beginnt Python mit der Ausführung der Schleife. Bei `message = input(prompt)` angekommen, zeigt Python die Eingabeaufforderung an und wartet auf die Benutzereingabe, die daraufhin in `message` gespeichert und ausgegeben wird. Danach prüft Python die Bedingung in der `while`-Anweisung erneut. Solange der Benutzer nicht das Wort `'quit'` eingibt, wird weiterhin die Eingabeaufforderung angezeigt. Erst wenn die Eingabe `'quit'` lautet, beendet Python die `while`-Schleife und das Programm:

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello everyone!
Hello everyone!

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello again.
Hello again.

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
quit
```

Das Programm funktioniert zwar schon ganz gut, allerdings gibt es das Wort `'quit'` ebenfalls aus, als sei es eine der zu wiederholenden Aussagen. Das können wir jedoch mit einem einfachen `if`-Test beheben:

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "

message = ""
while message != 'quit':
    message = input(prompt)

    if message != 'quit':
        print(message)
```

Jetzt prüft das Programm den Inhalt von `message` vor der Anzeige und gibt ihn nur dann aus, wenn es sich dabei nicht um den Beendigungswert handelt:

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello everyone!
Hello everyone!

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello again.
Hello again.

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
```

Flags

In dem vorstehenden Beispiel hat das Programm seine Aufgaben ausgeführt, solange eine bestimmte Bedingung wahr war. Was aber machen wir bei anspruchsvolleren Programmen, bei denen viele verschiedene Ereignisse dazu führen können, dass die Ausführung beendet wird? Das kann beispielsweise bei einem Spiel der Fall sein, wenn einem Spieler die Schiffe ausgehen, die Zeit abgelaufen ist oder alle Städte zerstört sind, die der Spieler hätte schützen müssen. Alle diese Bedingungen in einer `while`-Anweisung zu prüfen, wäre ziemlich kompliziert.

Damit ein Programm nur so lange läuft, bis eine von vielen Bedingungen nicht mehr wahr ist, können wir eine Variable definieren, die bestimmt, ob das Programm weiterlaufen soll oder nicht. Diese Variable wird als *Flag* bezeichnet und dient als Signal. Das Programm läuft, solange das Flag den Wert `True` hat, und wird beendet, sobald eines von vielen verschiedenen Ereignissen das Flag auf `False` setzt. Dadurch muss die `while`-Anweisung nur eine einzige Bedingung prüfen, nämlich den Zustand des Flags. Alle anderen Tests (mit denen wir feststellen, ob ein Ereignis eingetreten ist, aufgrund dessen das Flag auf `False` gesetzt werden muss) können wir dann sauber im Rest des Programms unterbringen.

Im Folgenden fügen wir dem Programm aus dem letzten Abschnitt ein Flag hinzu. Es trägt den Namen `active` (wobei Sie auch jeden beliebigen anderen Namen wählen können) und gibt an, ob das Programm weiterlaufen soll oder nicht:

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "

❶ active = True
❷ while active:
      message = input(prompt)

❸     if message == 'quit':
          active = False
❹     else:
          print(message)
```

Zu Anfang setzen wir die Variable `active` auf `True` (❶), damit das Programm im aktiven Zustand beginnt. Das macht die `while`-Anweisung einfacher, da wir den Vergleich der Eingabe dort herausgenommen und in einen anderen Teil des Programms verlagert haben. Die Schleife wird einfach ausgeführt, solange die Variable `active` den Wert `True` behält (❷).

In der `if`-Anweisung innerhalb der `while`-Schleife prüfen wir den Wert von `message`, nachdem der Benutzer eine Eingabe gemacht hat. Lautet diese Eingabe `'quit'` (❸), setzen wir `active` auf `False`, wodurch die `while`-Schleife beendet wird. Gibt der Benutzer einen beliebigen anderen Wert ein (❹), geben wir diesen aus.

Die Ausgabe ist identisch mit der aus der früheren Version des Programms, bei der die Überprüfung der Eingabe unmittelbar in der `while`-Anweisung erfolgte. Da wir jetzt ein Flag haben, um zu bestimmen, ob das Programm weiterlaufen soll oder nicht, können wir auf einfache Weise zusätzliche Tests (z. B. durch `elif`-Anweisungen) hinzufügen, um weitere Ereignisse abzudecken, durch die `active` auf `False` gesetzt werden soll. Das ist bei komplexeren Programmen praktisch, etwa bei Spielen, bei denen viele verschiedene Ereignisse zur Beendigung führen können. Setzt eines dieser Ereignisse das Flag auf `False`, so wird die Hauptschleife beendet. Daraufhin können Sie noch die Meldung *Game over!* sowie eine Möglichkeit dafür anzeigen, das Spiel erneut zu starten.

Eine Schleife mit break verlassen

Mit der Anweisung `break` können Sie eine `while`-Schleife unabhängig von irgendeiner Bedingung sofort abbrechen, ohne den restlichen Schleifencode auszuführen. Diese Anweisung steuert den Fluss des Programms. Sie können damit festlegen, welche Codezeilen ausgeführt werden und welche nicht.

Betrachten Sie als Beispiel das folgende Programm, das den Benutzer nach bereits besuchten Orten fragt. Wir können die `while`-Schleife abbrechen, indem wir `break` aufrufen, sobald der Benutzer den Wert `'quit'` eingibt.

cities.py

```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "

❶ while True:
    city = input(prompt)

    if city == 'quit':
        break
    else:
        print("I'd love to go to " + city.title() + "!")
```

Eine Schleife, die mit `while True` beginnt (❶), läuft endlos, sofern sie nicht durch eine `break`-Anweisung abgebrochen wird. In unserem Programm fordert die Schleife den Benutzer auf, die Namen von bereits besuchten Städten einzugeben. Das geht so lange, bis der Benutzer `'quit'` eingibt. In diesem Fall wird die `break`-Anweisung ausgeführt, die Python dazu veranlasst, die Schleife zu beenden:

```
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) New York
I'd love to go to New York!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) San Francisco
I'd love to go to San Francisco!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) quit
```

Hinweis

Die Anweisung `break` können Sie in Python in jeder Art von Schleife verwenden, also auch, um eine `for`-Schleife abzubrechen, die eine Liste oder ein Dictionary durchläuft.

Die Anweisung continue

Anstatt eine Schleife komplett abzubrechen, ohne den Rest des Codes auszuführen, können Sie mit der Anweisung `continue` auch zum Anfang der Schleife zurückspringen. Betrachten Sie als Beispiel die folgende Schleife, die von 1 bis 10 zählt, aber nur die ungeraden Zahlen aus diesem Bereich ausgibt:

counting.py

```
current_number = 0
while current_number < 10:
❶     current_number += 1
    if current_number % 2 == 0:
        continue

    print(current_number)
```

Zu Anfang setzen wir `current_number` auf 0. Da dies kleiner als 10 ist, tritt Python in die `while`-Schleife ein. Dort setzen wir `current_number` um 1 herauf (❶), sodass die Variable jetzt den Wert 1 hat. Die `if`-Anweisung schaut sich den Modulo von `current_number` und 2 an. Lautet das Ergebnis 0 (ist `current_number` also durch 2 teilbar), wird Python mithilfe der Anweisung `continue` angewiesen, den Rest der Schleife zu ignorieren und wieder zum Anfang zurückzuspringen. Ist der Wert der

Variablen dagegen nicht durch 2 teilbar, wird der Rest der Schleife ausgeführt und die Zahl ausgegeben:

```
1
3
5
7
9
```

Endlosschleifen vermeiden

Jede `while`-Schleife muss enden können, damit sie nicht endlos weiterläuft. Beispielsweise soll die folgende Schleife nur von 1 bis 5 zählen:

counting.py

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

Wenn Sie aber versehentlich die Zeile `x += 1` weglassen, läuft die Schleife endlos:

```
# Diese Schleife läuft endlos!
x = 1
while x <= 5:
    print(x)
```

Der Wert von `x` beginnt nach wie vor bei 1, ändert sich jetzt aber nicht mehr. Daher ergibt der Test `x <= 5` stets `True`, sodass die `while`-Schleife eine Folge von Einsen ausgibt und niemals endet:

```
1
1
1
1
-- schnipp! --
```

Allen Programmierern passiert es hin und wieder, dass sie eine Endlosschleife schreiben, vor allem, wenn die Beendigungsbedingungen etwas verzwickter sind. Wenn Ihr Programm in einer Endlosschleife hängen bleibt, drücken Sie Strg + C oder schließen Sie das Terminalfenster, in dem es ausgegeben wird.

Um ein solches Verhalten zu vermeiden, testen Sie alle `while`-Schleifen und vergewissern Sie sich, dass sie dann anhalten, wenn Sie es erwarten. Wenn das Programm enden soll, sobald der Benutzer einen bestimmten Wert eingibt, dann führen Sie es aus und geben diesen Wert ein. Bricht es daraufhin nicht ab, schauen Sie sich an, wie das Programm den Beendigungswert verarbeitet. Stellen Sie sicher,

dass mindestens ein Teil des Programms die Schleifenbedingung auf `False` setzen kann, oder sorgen Sie dafür, dass es eine `break`-Anweisung erreicht.

Hinweis

Einige Editoren, darunter auch Sublime Text, verwenden eingebettete Ausgabefenster. Das kann es schwierig machen, eine Endlosschleife abzubrechen, sodass Sie den Editor schließen müssen, um die Schleife zu beenden.

Probieren Sie es selbst aus!

7-4 Pizzabeläge: Schreiben Sie eine Schleife, die den Benutzer so lange auffordert, Pizzabeläge einzugeben, bis er einen Beendigungswert eingibt. Geben Sie zu jedem Belag die Meldung aus, dass er der Pizza hinzugefügt wird.

7-5 Eintrittskarten: Ein Kino staffelt seine Eintrittspreise nach dem Alter der Besucher. Eintrittskarten für Kinder unter 3 Jahren sind umsonst. Für Personen zwischen 3 und 12 Jahren kosten sie 10 $, für Personen über 12 Jahren 15 $. Schreiben Sie eine Schleife, in der die Besucher nach ihrem Alter gefragt werden und dann den Preis ihrer Eintrittskarte erfahren.

7-6 Drei Arten der Beendigung: Schreiben Sie unterschiedliche Versionen des Programms aus Übung 7-4 oder 7-5 mit den drei folgenden Vorgehensweisen:

- Die Schleife wird aufgrund einer Bedingung in der `while`-Anweisung beendet.
- Eine Flagvariable steuert, wie lange die Schleife ausgeführt wird.
- Die Schleife wird mithilfe einer `break`-Anweisung abgebrochen, wenn der Benutzer einen Beendigungswert eingibt.

7-7 Endlosschleife: Schreiben Sie eine Endlosschleife und führen Sie sie aus. (Um die Schleife abzubrechen, drücken Sie Strg + C oder schließen das Ausgabefenster.)

while-Schleifen für Listen und Dictionaries

Bis jetzt haben wir immer nur eine Benutzerinformation auf einmal verarbeitet: Wir haben eine Eingabe entgegengenommen und danach diese Eingabe oder eine Antwort darauf ausgegeben. Bei der nächsten Ausführung der `while`-Schleife haben wir dann einen weiteren Eingabewert empfangen und darauf reagiert. Um auch bei mehreren Benutzern und mehreren Informationen den Überblick zu bewahren, müssen wir zusammen mit unseren `while`-Schleifen auch Listen und Dictionaries einsetzen.

Mit einer `for`-Schleife können Sie eine Liste durchlaufen, aber Sie sollten Listen nicht in einer solchen Schleife bearbeiten, da Python sonst Schwierigkeiten hat,

den Überblick über die Elemente zu behalten. Zum Bearbeiten von Listen setzen Sie eine `while`-Schleife ein. Mit einer Kombination aus `while`-Schleifen, Listen und Dictionaries können Sie viele Eingaben erfassen, speichern und gliedern, um sie zu untersuchen oder später einen Bericht darüber zu erstellen.

Elemente von einer Liste in eine andere verschieben

Nehmen wir an, Sie haben eine Liste neu registrierter, aber noch nicht bestätigter Benutzer einer Website. Wie können Sie sie nach der Verifizierung in eine Liste bestätigter Benutzer verschieben? Eine Möglichkeit besteht darin, eine `while`-Schleife einzusetzen. Der Code dafür kann wie folgt aussehen:

confirmed_users.py

```
    # Beginnt mit einer Liste der zu überprüfenden Benutzer
    # und einer leeren Liste der bereits bestätigten Benutzer
❶   unconfirmed_users = ['alice', 'brian', 'candace']
    confirmed_users = []

    # Prüft alle Benutzer, bis keine unbestätigten mehr vorhanden sind.
    # Verschiebt jeden bestätigten Benutzer in die entsprechende Liste.
❷   while unconfirmed_users:
❸       current_user = unconfirmed_users.pop()

        print("Verifying user: " + current_user.title())
❹       confirmed_users.append(current_user)

    # Zeigt alle bestätigten Benutzer an.
    print("\nThe following users have been confirmed:")
    for confirmed_user in confirmed_users:
        print(confirmed_user.title())
```

Wir beginnen mit einer Liste der unbestätigten Benutzer (Alice, Brian und Candace) und einer leeren Liste für die bestätigten (❶). Die `while`-Schleife bei ❷ läuft, solange die Liste `unconfirmed_users` nicht leer ist. Innerhalb dieser Schleife entfernt die Funktion `pop()` bei ❸ einen nicht bestätigten Benutzer nach dem anderen vom Ende der Liste `unconfirmed_users`, der anschließend in `current_user` gespeichert und dann bei ❹ zur Liste `confirmed_users` hinzugefügt wird. Da Candace am Ende von `unconfirmed_users` steht, wird ihr Name als erster auf diese Weise verarbeitet, danach Brian und schließlich Alice.

Den Verifizierungsvorgang simulieren wir dadurch, dass wir eine Überprüfungsmeldung ausgeben und den Namen zur Liste `confirmed_users` hinzufügen. Im Verlauf des Programms wird die Liste der bestätigten Benutzer immer länger, während die mit den unbestätigten Benutzern schrumpft. Wenn letztere leer ist, wird die Schleife beendet und die Liste der bestätigten Benutzer ausgegeben:

```
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice
```

Alle Vorkommen eines Wertes aus einer Liste entfernen

In Kapitel 3 haben Sie die Funktion `remove()` kennengelernt, mit der Sie einen bestimmten Wert aus einer Liste entfernen können – allerdings nur das erste Vorkommen. Was aber tun wir, wenn wir alle Vorkommen dieses Wertes löschen wollen?

In der folgenden Liste von Haustieren kommt der Wert `'cat'` mehrmals vor. Um alle diese Einträge zu entfernen, führen wir eine `while`-Schleife aus, bis `'cat'` nicht mehr in der Liste vorhanden ist:

pets.py

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

Nachdem wir die Liste ausgegeben haben, startet Python die `while`-Schleife, da der Wert `'cat'` mindestens einmal in der Liste vorkommt. In der Schleife entfernt Python das erste Vorkommen von `'cat'` und kehrt dann zu der `while`-Zeile zurück. Wenn sich immer noch `'cat'`-Einträge auf der Liste befinden, wird die Schleife erneut ausgeführt. So werden nach und nach alle Vorkommen von `'cat'` entfernt, bis der Wert nicht mehr in der Liste vorhanden ist. An dieser Stelle angekommen, beendet Python die Schleife und gibt die Liste erneut aus:

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

Ein Dictionary mit Benutzereingaben füllen

Bei jedem Durchlauf einer `while`-Schleife können Sie nach so vielen Eingaben fragen, wie Sie wollen. In dem folgenden Umfrageprogramm bitten wir die Benutzer darum, Ihren Namen und die Antwort auf eine Frage anzugeben. Die erfassten Daten speichern wir in einem Dictionary, da wir die Antworten jeweils dem Benutzer zuordnen möchten:

mountain_poll.py

```
responses = {}

# Richtet ein Flag ein, um anzugeben, dass die Umfrage aktiv bleiben soll.
polling_active = True

while polling_active:
    # Bittet den Benutzer um Eingabe seines Namens und der Antwort.
❶   name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to climb someday? ")

    # Speichert die Antwort im Dictionary:
❷   responses[name] = response

    # Fragt, ob noch jemand an der Umfrage teilnehmen wird.
❸   repeat = input("Would you like to let another person respond? (yes/no) ")
    if repeat == 'no':
        polling_active = False

# Die Umfrage ist abgeschlossen, die Ergebnisse werden angezeigt.
print("\n--- Poll Results ---")
❹ for name, response in responses.items():
    print(name + " would like to climb " + response + ".")
```

Zu Anfang des Programms wird das leere Dictionary `responses` eingerichtet und das Flag `pooling_active` gesetzt, das angibt, ob die Umfrage aktiv bleiben soll. Solange dieses Flag `True` ist, führt Python den Code in der `while`-Schleife aus.

Innerhalb der Schleife werden die Benutzer aufgefordert, ihren Benutzernamen und einen Berg zu nennen, den Sie gern besteigen möchten (❶). Diese Informationen werden im Dictionary `responses` gespeichert (❷). Daraufhin wird der Benutzer gefragt, ob die Umfrage weitergeführt werden soll oder nicht (❸). Gibt er `yes` an, wird die `while`-Schleife erneut ausgeführt. Bei `no` dagegen wird das Flag `polling_active` auf `False` gesetzt, sodass die `while`-Schleife beendet wird. Der Codeblock am Ende (❹) zeigt die Ergebnisse der Umfrage an.

Wenn Sie dieses Programm ausführen, erhalten Sie eine Ausgabe wie die folgende:

```
What is your name? Eric
Which mountain would you like to climb someday? Denali
Would you like to let another person respond? (yes/no) yes

What is your name? Lynn
Which mountain would you like to climb someday? Devil's Thumb
Would you like to let another person respond? (yes/no) no

--- Poll Results ---
Lynn would like to climb Devil's Thumb.
Eric would like to climb Denali.
```

Probieren Sie es selbst aus!

7-8 Sandwiches: Erstellen Sie die Liste `sandwich_orders`, die Sie mit den Namen verschiedener Sandwichvariationen füllen, sowie die Liste `finished_sandwiches`. Durchlaufen Sie die Liste der Bestellungen, geben Sie für jede davon eine Meldung wie *I made your tuna sandwich* aus und verschieben Sie das betreffende Element in die Liste der fertigen Sandwiches. Geben Sie abschließend eine Liste aller zubereiteten Sandwiches aus.

7-9 Kein Pastrami: Führen Sie in der Liste `sandwich_orders` aus Übung 7-8 mindestens dreimal das Sandwich `'pastrami'` auf. Fügen Sie am Anfang des Programms Code hinzu, um die Meldung auszugeben, dass Pastrami ausgegangen ist, und entfernen Sie mithilfe einer `while`-Schleife alle Vorkommen von `'pastrami'` von `sandwich_orders`. Sorgen Sie dafür, dass keinerlei Sandwiches mit Pastrami auf der Liste `finished_sandwiches` landen.

7-10 Traumurlaub: Schreiben Sie ein Programm, das die Benutzer nach Zielen für einen Traumurlaub fragt. Schreiben Sie dazu eine Eingabeaufforderung wie: »If you could visit one place in the world, where would you go?« Fügen Sie einen Codeblock hinzu, der die Ergebnisse der Umfrage ausgibt.

Zusammenfassung

In diesem Kapitel habe Sie gelernt, wie Sie Benutzern mit der Funktion `input()` erlauben, selbst Informationen in Ihr Programm einzugeben, wie Sie solche Eingaben sowohl in Text- als auch in numerischer Form abrufen und wie Sie `while`-Schleifen einsetzen, um Programme so lange auszuführen, wie die Benutzer das wollen. Sie haben auch mehrere Möglichkeiten gesehen, um den Fluss einer `while`-Schleife zu steuern – nämlich mithilfe eines Flags oder der Anweisungen `break` und `continue`. Des Weiteren haben Sie erfahren, wie Sie mithilfe einer `while`-Schleife Elemente von einer Liste in eine andere verschieben und alle Vorkommen eines Wertes aus einer Liste entfernen. Sie wissen jetzt auch, wie Sie `while`-Schleifen und Dictionaries kombinieren.

In Kapitel 8 geht es um *Funktionen*. Damit können Sie Ihre Programme in kleinere Teile gliedern, die jeweils eine ganz bestimmte Aufgabe erfüllen. Eine Funktion können Sie so oft aufrufen, wie Sie wollen, und Sie können sie auch in eigenen Dateien speichern. Mithilfe von Funktionen können Sie effizienteren Code schreiben, der sich leichter korrigieren und pflegen sowie in anderen Programmen wiederverwenden lässt.