3

Zahlen raten

In diesem Kapitel schreiben Sie ein Zahlenratespiel (»Guess The Number«). Der Computer denkt sich eine Geheimzahl von 1 bis 20 aus und bittet den Benutzer, sie zu erraten. Nach jedem Versuch teilt der Computer dem Benutzer mit, ob die eingegebene Zahl zu hoch oder zu niedrig war. Der Benutzer gewinnt, wenn er die Zahl mit sechs Versuchen erraten kann.

Dieses Spiel ist zum Lernen sehr gut geeignet, da das Programm dafür sehr kurz ist, aber viele Programmierprinzipien abdeckt. So lernen Sie dabei, wie Sie Werte in einen anderen Datentyp umwandeln und wann Sie das tun müssen. Da es sich bei dem Programm um ein Spiel handelt, werden wir den Benutzer von nun an den *Spieler* nennen.

Themen in diesem Kapitel

- `import`-Anweisungen
- Module
- Die Funktion `randint()`
- `for`-Anweisungen
- Blöcke
- Die Funktionen `str()`, `int()` und `float()`
- Boolesche Werte
- Vergleichsoperatoren
- Bedingungen
- Der Unterschied zwischen `=` und `==`
- `if`-Anweisungen
- `break`-Anweisungen

Ein Beispieldurchlauf des Zahlenratespiels

Das folgende Listing zeigt, was ein Spieler angezeigt bekommt, wenn er das Zahlenratespiel ausführt. Die Eingaben des Spielers sind dabei durch Fettdruck gekennzeichnet.

```
Hello! What is your name?
Albert
Well, Albert, I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too high.
Take a guess.
2
Your guess is too low.
Take a guess.
4
Good job, Albert! You guessed my number in 3 guesses!
```

Der Quellcode für das Zahlenratespiel

Klicken Sie auf *File* > *New Windows*, um ein neues Dateieditorfenster zu öffnen. Geben Sie den folgenden Quellcode in das leere Fenster ein und speichern Sie ihn als *guess.py*. Führen Sie das Programm anschließend mit F5 aus.

Achten Sie bei der Eingabe genau auf die Leerzeichen. Einige Zeilen müssen um vier oder acht Leerzeichen eingerückt werden.

Falls Sie eine Fehlermeldung erhalten, vergleichen Sie den eingegebenen Code mit dem Code aus dem Buch. Dazu können Sie das Diff Tool auf *https://www.nostarch.com/inventwithpython#diff* verwenden.

guess.py

```
 1. # Dies ist ein Zahlenratespiel.
 2. import random
 3.
 4. guessesTaken = 0
 5.
 6. print('Hello! What is your name?')
 7. myName = input()
 8.
 9. number = random.randint(1, 20)
10. print('Well, ' + myName + ', I am thinking of a number between 1
          and 20.')
11.
12. for guessesTaken in range(6):
13.     print('Take a guess.') # Vier Leerzeichen vor "print"
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
18.         print('Your guess is too low.') # Acht Leerzeichen vor "print"
19.
20.     if guess > number:
21.         print('Your guess is too high.')
22.
23.     if guess == number:
24.         break
25.
26. if guess == number:
27.     guessesTaken = str(guessesTaken + 1)
28.     print('Good job, ' + myName + '! You guessed my number in ' +
          guessesTaken + ' guesses!')
29.
30. if guess != number:
31.     number = str(number)
32.     print('Nope. The number I was thinking of was ' + number + '.')
```

Das Modul random importieren

Sehen wir uns zunächst die beiden ersten Zeilen des Programms an:

```
1. # Dies ist ein Zahlenratespiel.
2. import random
```

Die erste Zeile ist ein Kommentar, wie Sie ihn schon in Kapitel 2 kennengelernt haben. Wie Sie bereits wissen, ignoriert Python alles, was auf das Zeichen `#` folgt. Der Kommentar gibt lediglich an, wozu das Programm gut ist.

Bei der zweiten Zeile handelt es sich um eine `import`-Anweisung. Wie Sie schon gelernt haben, sind Anweisungen Befehle, die irgendeine Aktion durchführen, im Gegensatz zu Ausdrücken aber nicht zu einem Wert ausgewertet werden. Sie kennen bereits die Zuweisungsanweisung, die einen Wert in einer Variablen speichert.

Python wird mit vielen vorgefertigten Funktionen geliefert, von denen sich einige jedoch in separaten Programmen befinden, nämlich in sogenannten *Modulen*. Um diese Funktionen zu nutzen, müssen Sie das betreffende Modul mit einer `import`-Anweisung in Ihr Programm importieren.

Zeile 2 importiert das Modul `random`, damit das Programm die darin enthaltene Funktion `randint()` aufrufen kann. Diese Funktion erzeugt die Zufallszahl, die der Spieler raten soll.

Nach dem Import des Moduls erstellen Sie in Zeile 4 als Nächstes die neue Variable `guessesTaken`, sodass sie später in dem Programm verwendet werden kann:

```
4. guessesTaken = 0
```

Diese Variable enthält die Anzahl der Rateversuche, die der Spieler schon durchgeführt hat. Da der Spieler an dieser Stelle im Programm noch gar nichts geraten hat, speichern Sie zunächst den Integer `0` in dieser Variablen.

```
6. print('Hello! What is your name?')
7. myName = input()
```

Die Zeilen 6 und 7 sind identisch mit den entsprechenden Zeilen aus dem Hello-World-Programm in Kapitel 2. Programmierer greifen oft auf Code aus anderen Programmen zurück, um sich selbst Arbeit zu ersparen.

Zeile 6 ruft die Funktion `print()` auf. Wie Sie bereits wissen, ist eine Funktion eine Art Miniprogramm innerhalb Ihres Programms. Wird diese Funktion aufgerufen, so wird dieses Miniprogramm ausgeführt. Der Code von `print()` zeigt das Stringargument, das Sie übergeben, auf dem Bildschirm an.

Zeile 7 ruft den vom Spieler eingegebenen Namen ab und speichert ihn in der Variablen `myName`. Bei diesem String muss es sich nicht unbedingt um den wirklichen Namen des Spielers handeln; er enthält einfach das, was der Spieler eingegeben hat. Computer denken nicht, sondern befolgen stur die Befehle, die man ihnen gegeben hat.

Zufallszahlen mit der Funktion random.randint() erzeugen

Nun können Sie die Funktion aus dem Modul `random` verwenden, um die Geheimzahl des Computers zu erzeugen:

```
9. number = random.randint(1, 20)
```

Zeile 9 ruft die Funktion `randint()` auf und speichert deren Rückgabewert in `number`. Wie Sie bereits wissen, lassen sich Funktionsaufrufe in Ausdrücken verwenden, da sie zu einem Wert ausgewertet werden können.

Da die Funktion `randint()` vom Modul `random` bereitgestellt wird, müssen Sie sie mit `random.randint()` aufrufen (vergessen Sie nicht den Punkt!), damit Python weiß, in welchem Modul es danach Ausschau halten muss.

`randint()` gibt einen zufälligen Integer aus dem Bereich zwischen den beiden übergebenen Integerargumenten (und einschließlich dieser Argumente) zurück. In den Klammern hinter dem Funktionsnamen werden dazu in Zeile 9 die Zahlen `1` und `20` durch ein Komma getrennt übergeben. Die zurückgegebene Zufallszahl wird in der Variablen `number` gespeichert. Dies ist die Geheimzahl, die der Spieler erraten muss.

Um sich das genauer anzusehen, kehren Sie vorübergehend zur interaktiven Shell zurück. Geben Sie dort **import random** ein, um das Modul zu importieren, und dann **random.randint(1, 20)**, um zu beobachten, wie der Funktionsaufruf ausgewertet wird. Sie werden feststellen, dass er eine zufällige Zahl aus dem Bereich von 1 bis 20 zurückgibt. Wenn Sie den Funktionsaufruf erneut eingeben, gibt er einen weiteren Integer zurück. Die Funktion `randint()` gibt jedes Mal eine Zufallszahl zurück, ebenso wie der Wurf eines Würfels. Wenn Sie die folgenden Befehle in die interaktive Shell eingeben, werden Sie andere Ergebnisse sehen als hier im Buch, denn schließlich sind sie ja zufällig.

```
>>> import random
>>> random.randint(1, 20)
12
>>> random.randint(1, 20)
18
>>> random.randint(1, 20)
3
```

```
>>> random.randint(1, 20)
18
>>> random.randint(1, 20)
7
```

Probieren Sie auch andere Zahlenbereiche aus, indem Sie die Argumente ändern. Wenn Sie beispielsweise **random.randint(1, 4)** eingeben, erhalten Sie nur Integer von 1 bis 4 (einschließlich). Mit **random.randint(1000, 2000)** erzeugen Sie Integer aus dem Bereich von 1000 bis 2000:

```
>>> random.randint(1, 4)
3
>>> random.randint(1000, 2000)
1294
```

Durch leichtere Änderungen am Code können Sie das Verhalten des Spiels ändern. Im ursprünglichen Code verwenden wir einen Integer aus dem Bereich von 1 bis 20:

```
 9. number = random.randint(1, 20)
10. print('Well, ' + myName + ', I am thinking of a number between 1 and
          20.')
```

Versuchen Sie, den Bereich in `(1, 100)` zu ändern:

```
 9. number = random.randint(1, 100)
10. print('Well, ' + myName + ', I am thinking of a number between 1 and
          100.')
```

Jetzt denkt sich der Computer eine Ganzzahl von 1 bis 100 statt von 1 bis 20 aus. Durch die Änderung von Zeile 9 erweitern Sie den Bereich, aus dem die Zufallszahl stammt. Denken Sie aber daran, auch Zeile 10 anzupassen, damit das Programm dem Spieler den neuen Bereich mitteilt und nicht mehr den alten.

Die Funktion `randint()` können Sie überall dort einsetzen, wo Sie in Ihren Spielen Zufallszahlen benötigen, und das ist in vielen Spielen der Fall. (Denken Sie nur einmal daran, in wie vielen Brettspielen Würfel verwendet werden.)

Den Spieler begrüßen

Nachdem der Computer der Variablen `number` eine Zufallszahl zugewiesen hat, begrüßt er den Spieler:

```
10. print('Well, ' + myName + ', I am thinking of a number between 1 and
          20.')
```

Die Funktion `print()` in Zeile 10 begrüßt den Spieler mit Namen und teilt ihm mit, dass sich der Computer eine Zufallszahl ausgedacht hat.

Auf den ersten Blick sieht es so aus, als hätte die Funktion mehr als ein Stringargument. Schauen Sie sich die Zeile aber genauer an. Die beiden Verkettungsoperatoren kombinieren die drei Strings zu einem einzigen, der dann als Argument an `print()` übergeben wird. Die Kommas befinden sich innerhalb der Anführungszeichen und gehören damit zu den Strings.

Flusssteuerungsanweisungen

In den vorherigen Kapiteln wurden die Befehle bei der Programmausführung einer nach dem anderen in der vorliegenden Reihenfolge abgearbeitet. Mit den Anweisungen `for`, `if`, `else` und `break` können Sie die Befehle jedoch auch nach bestimmten Bedingungen in einer Schleife ausführen lassen oder einzelne Anweisungen überspringen. Solche Anweisungen werden als *Flusssteuerungsanweisungen* bezeichnet, da sie den Fluss der Programmausführung ändern.

Code in Schleifen wiederholen

Zeile 12 enthält die Anweisung `for`, die den Beginn einer `for`-Schleife kennzeichnet:

```
12. for guessesTaken in range(6):
```

In *Schleifen* können Sie Code wiederholt ausführen. Zeile 12 sorgt dafür, dass der nachfolgende Code sechsmal ausgeführt wird. Eine `for`-Anweisung beginnt mit dem Schlüsselwort `for`. Darauf folgen ein neuer Variablenname, das Schlüsselwort `in`, der Aufruf der Funktion `range()`, der die Anzahl der Schleifendurchläufe vorgibt, und ein Doppelpunkt. Damit Sie sicher mit Schleifen arbeiten können, müssen wir uns jedoch zunächst mit einigen anderen Grundprinzipien vertraut machen.

Blöcke

Mehrere Codezeilen können zu einem *Block* gruppiert werden. Am Anfang jeder Codezeile in dem Block stehen mindestens genauso viele Leerzeichen wie am Anfang der ersten Zeile des Blocks. Anhand der Anzahl der Leerzeichen am Anfang der Zeilen – also der *Einrückung* der Zeilen – können Sie erkennen, wo ein Block anfängt und wo er endet.

Um einen Block zu beginnen, verwenden Python-Programmierer gewöhnlich vier *zusätzliche* Leerzeichen. Alle folgenden Zeilen, die um den gleichen Betrag von Leerzeichen eingerückt sind, gehören zu dem Block. Der Block endet, wenn

eine Codezeile die *gleiche Einrückung* aufweist, die vor dem Beginn des Blocks herrschte. Ein Block kann jedoch auch weitere Blöcke einschließen. In Abbildung 3–1 sind die Blöcke umrandet und nummeriert.

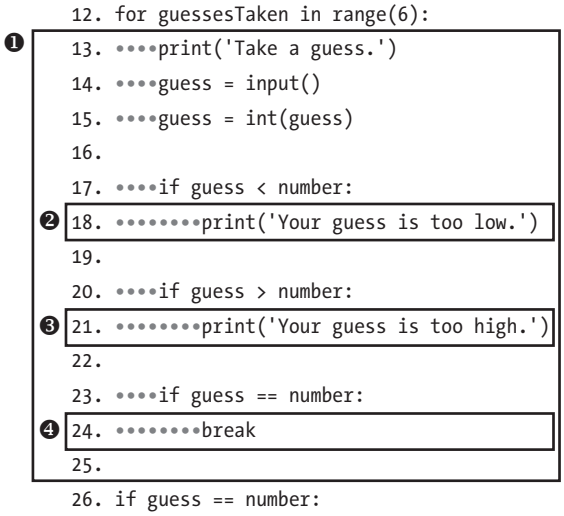

Abb. 3–1 *Blöcke und ihre Einrückungen. Die grauen Punkte stehen für Leerzeichen.*

In Abbildung 3–1 ist Zeile 12 nicht eingerückt und gehört daher nicht zu einem Block. Zeile 13 dagegen beginnt mit vier Leerzeilen. Da sie weiter eingerückt ist als die vorherige, beginnt hier ein neuer Block. Jede folgende Zeile, die die gleiche oder eine noch stärkere Einrückung aufweist, wird als Bestandteil von Block ❶ angesehen. Trifft Python dagegen auf eine Zeile mit einer geringeren Einrückung als in der ersten Zeile des Blocks, dann endet der Block. Leerzeilen werden dabei ignoriert.

Zeile 18 weist eine Einrückung um acht Leerzeichen auf, weshalb hier Block ❷ beginnt. Dieser Block befindet sich *innerhalb* von Block ❶. Die nächste Zeile, Nr. 20, ist wiederum nur vier Zeilen eingerückt. Da die Einrückung wieder abgenommen hat, ist der Block ❷ damit beendet, doch da Zeile 20 die gleich Einrückung aufweist wie Zeile 13, befindet sie sich nach wie vor in Block ❶.

Zeile 21 erhöht die Einrückung wieder auf acht Leerzeichen, sodass ein neuer Block innerhalb des Blocks beginnt, nämlich Block ❸. In Zeile 23 verlassen wir Block ❸ wieder, und in Zeile 24 beginnt der letzte verschachtelte Block (Block ❹). Sowohl Block ❶ als auch ❹ enden in Zeile 24.

for-Schleifen

Die `for`-Anweisung kennzeichnet den Beginn einer Schleife. Schleifen dienen dazu, Code wiederholt auszuführen. Nach dem Erreichen einer `for`-Anweisung fährt die Ausführung mit dem darauffolgenden Block fort, doch sobald der gesamte Code in diesem Block abgearbeitet ist, kehrt die Ausführung wieder zum Anfang des Blocks zurück, sodass der Code erneut abgearbeitet wird.

Um das zu veranschaulichen, geben Sie Folgendes in die interaktive Shell ein:

```
>>> for i in range(3):
    print('Hello! i is set to', i)


Hello! i is set to 0
Hello! i is set to 1
Hello! i is set to 2
```

Nachdem Sie `for i in range(3):` eingegeben und die Eingabetaste gedrückt haben, hat die Shell nicht wie üblich die Eingabeaufforderung `>>>` angezeigt, da sie nun erwartet, dass Sie einen Codeblock hinzufügen. Drücken Sie nach der Eingabe des letzten Befehls ein weiteres Mal die Eingabetaste, damit die Shell weiß, dass der Codeblock komplett ist. (Das gilt nur für die Arbeit in der interaktiven Shell. Wenn Sie im Dateieditor *.py*-Dateien schreiben, müssen Sie keine Leerzeile eingeben.)

Die `for`-Schleife in *guess.py* sieht wie folgt aus:

```
12. for guessesTaken in range(6):
13.     print('Take a guess.') # Vier Leerzeichen vor "print"
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
18.         print('Your guess is too low.') # Acht Leerzeichen vor "print"
19.
20.     if guess > number:
21.         print('Your guess is too high.')
22.
23.     if guess == number:
24.         break
25.
26. if guess == number:
```

Der `for`-Block beginnt bei der `for`-Anweisung in Zeile 12. Die erste Zeile hinter dem Block ist Zeile 26.

In einer `for`-Anweisung steht immer ein Doppelpunkt hinter der Bedingung. Bei Anweisungen, die mit einem Doppelpunkt enden, wird immer ein neuer Block in der folgenden Zeile erwartet.