|  |  |
| --- | --- |
|  | Kapitel 4 |
| Rate die Zahl |

In diesem Kapitel lernst du:

* import Kommandos
* Module
* while Kommandos
* Bedingungen
* Blöcke
* Boolean-Datentypen
* Vergleichsoperatoren
* Den Unterschied zwischen = und ==
* if Kommandos
* Das break Kommando
* Die str()and int() und float() Funktionen
* Die random.randint() Funktion

In diesem Kapitel wirst du ein "Rate die Zahl" Spiel schreiben. Der Computer wird an eine Zufallzahl zwischen 1 und 20 denken und dich bitten, sie zu erraten. Der Computer wird dir bei jedem Tipp, den du abgibst, sagen ob er zu hoch oder zu niedrig war. Du gewinnst wenn du die Zahl innerhalb von sechs Versuchen erraten kannst.

"Rate die Zahl" ist interessant zu programmieren weil es Zufallszahlen, Schleifen und Benutzereingaben in einem ziemlich kurzen Programm verwendet. Du wirst lernen, wie man Werte in unterschiedliche Datentypen umwandelt (und warum das notwendig ist). Weil unser Programm ein Spiel ist, nennen wir den Benutzer Spieler, aber "Benutzer" wäre auch ok.

## Beispiel Lauf zu Rate die Zahl

So sieht das Programm für den Spieler aus, wenn es ausgeführt wird. Der Text den der Spieler eintippt ist **fett geschrieben**.

Hallo! Was ist dein Name?

Albert

Also, Albert, ich denke an eine Zahl zwischen 1 und 20.

Los, rate.

10

Dein Tipp ist zu hoch.

Los, rate.

2

Dein Tipp ist zu niedrig.

Los, rate.

4

Gut gemacht, Albert! Du hast meine Zahl in 3 Zügen erraten.

## Der Quellcode von “Rate die Zahl”

Öffne ein neues Dateieditorfenster indem auf **File** ► **New Window** (Datei ► Neues Fenster) klickst. Tippe in dem neu erschienen leeren Fenster den Quellcode ein und speichere die Datei dann unter zahlenRaten.py ab. Drücke **F5** um das Programm auszuführen. Wenn du den Programmcode eintippst muss du besonders auf die Leerzeichen am Beginn der Zeilen achten. Manche Zeilen haben vier und einige sogar acht Leerzeichen am Zeilenanfang.

**WICHTIGER HINWEIS!** Die Programme in diesem Buch können nur in Python 3 ausgeführt werden, nicht in Python 2. Wenn IDLE startet sollte es so etwas wie "Python 3.4.2" oben anzeigen. Du kannst Python 2 und Python 3 zur selben Zeit auf deinem Computer installiert haben. Um Python 3 herunter zu laden geh zu https://python.org/download/.

Wenn du nach dem Eintippen des Codes Fehlermeldungen bekommst, verwende das Online diff Werkzeug unter http://invpy.com/de/diff/zahlenRaten um deinen Code mit dem Code aus dem Buch zu vergleichen.

zahlenRaten.py

1. # Das ist ein Zahlenratespiel.

2. import random

3.

4. abgegebenTipps = 0

5.

6. print('Hallo! Was ist dein Name?')

7. meinName = input()

8.

9. zahl = random.randint(1, 20)

10. print('Also, ' + meinName + ', ich denke an eine Zahl zwischen 1 und 20.')

11.

12. while abgegebenTipps < 6:

13. print('Los, rate.') # Vor print sind vier Leerzeichen.

14. tipp = input()

15. tipp = int(tipp)

16.

17. abgegebenTipps = abgegebenTipps + 1

18.

19. if tipp < zahl:

20. print('Dein Tipp ist zu niedrig.') # Hier sind acht Leerzeichen vor print.

21.

22. if tipp > zahl:

23. print('Dein Tipp ist zu hoch.')

24.

25. if tipp == zahl:

26. break

27.

28. if tipp == zahl:

29. abgegebenTipps = str(abgegebenTipps)

30. print('Gut gemacht, ' + meinName + '! Du hast meine Zahl in ' + abgegebenTipps + ' Zügen erraten!')

31.

32. if tipp != zahl:

33. zahl = str(zahl)

34. print('Nene. Die Nummer an die ich gedacht habe war ' + zahl)

Schauen wir uns den Code Zeile für Zeile an um zu verstehen wie das Programm funktioniert.

## import Befehle

1. # Das ist ein Zahlenratespiel.

2. import random

Die erste Zeile ist ein Kommentar. Python ignoriert alles nach dem # Zeichen. Diese Zeile dient nur dazu, uns daran zu erinnern was das Programm macht.

Zeile Zwei ist ein import Befehl. Befehle sind Instruktionen die eine Handlung durchführen aber nicht wie Ausdrücke ausgewertete werden können. Wir haben bereits Befehle kennengelernt: Zuweisungsbefehle speichern einen Wert in einer Variable.

Python bringt viele Funktionen mit, ein großer Teil davon befindet sich aber in getrennten Programmen die Module genannt werden. Um diese Funktionen zu verwenden musst du die zugerhörigen Module mit einem import Befehl in dein Programm importieren.

Zeile 2 importiert das modul mit dem Namen random (engl. zufällig) damit das Programm random.randint() aufrufen kann.Diese Funktion liefert die Zufallszahlen die der Benutzer erraten muss.

4. abgegebenTipps = 0

Line 4 creates a new variable named guessesTaken. You’ll store the number of guesses the player has made in this variable. Since the player hasn’t made any guesses at this point in the program, store the integer 0 here.

6. print('Hallo! Was ist dein Name?')

7. meinName = input()

Zeilen 6 und 7 sind identisch mit dem Hallo Welt-Programm aus Kapitel 3. Programmierer verwenden Programmcode aus alten Programmen oft wieder, um sich Arbeit zu sparen.

Zeile 6 ruft die print()-Funktion auf. Wir erinnern uns, dass ein Funktionsaufruf wie ein Miniprogramm in unserem Programm ist. Wenn ein Programm eine Funktion aufruft, dann führt es dieses Miniprogramm aus. Der Code innerhalb der print()-Funktion zeigt das ihr übergebene Zeichenketten-Argument auf dem Bildschirm an.

Zeile 7 lässt den Benutzer seinen Name eingeben und speichert die Eingabe in der Variable meinName. (Die Zeichenkette muss nicht wirklich der Name des Spielers sein, sondern ist einfach genau die Zeichenkette, die der Spieler eintippt. Computer sind dumm und folgen einfach stur deinen Anweisungen.)

## Die random.randint()-Funktion

9. zahl = random.randint(1, 20)

Zeile 9 ruft eine neue Funktion mit dem Namen randint() auf und speichert den Rückgabewert in zahl ab. Funktionsaufrufe können Teil eines Ausdruckes sein, weil sie zu einem Wert ausgewerteten werden.

Die randint()-Funktion wird vom random-Modul zur Verfügung gestellt. Deshalb muss ihrem Aufruf random vorangestellt werden (vergiss den Punkt nicht!). Dies teilt Python mit, dass die Funktion randint() im Modul random zu finden ist.

Die randint()-Funktion wird eine Zufallszahl zwischen und einschließlich der beiden übergebenen Ganzzahlen-Argumente zurückgeben. Zeile 9 übergibt die Ganzzahlen 1 und 20 an randint(), indem sie von einem Komma getrennt in die Klammern hinter dem Funktionsnamen geschrieben werden. Die Zufallszahl, die randint() zurückgibt, wird in einer Variable namens zahl gespeichert; das ist die geheime Zahl, die der Spieler erraten muss.

Wechsle für einen Moment in die interaktive Konsole und gib import random ein, um das random-Modul zu importieren. Dann tippe random.randint(1, 20) und drücke die Eingabetaste, um zu sehen, was der Funktionsaufruf zurückgibt. Er wird eine Ganzzahl von 1 bis 20 zurückgeben. Wenn du die Eingabe wiederholst, wird die Funktion wahrscheinlich eine andere Zahl zurückgeben. Die randint()-Funktion gibt jedes Mal eine zufällige Zahl aus dem übergebenen Wertebereich zurück, ähnlich wie ein Würfel bei jedem Wurf auf einer zufälligen Zahl landet.

>>> 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

Verwende die randint()-Funktion, um deinem Spiel ein zufälliges Element hinzuzufügen. Wir werden das Zufallselement noch in vielen weiteren Spielen verwenden (denke daran, wie viele Brettspiele Würfel benutzen).

Du kannst den Wertebereich ändern, indem du verschiedene Argumente übergibst. Probiere zum Beispiel random.randint(1, 4) einzugeben, um nur Ganzzahlen von 1 bis 4 (einschließlich 1 und 4) zu erhalten. Oder gib random.randint(1000, 2000) ein, um Ganzzahlen zwischen 1000 und 2000 zu erhalten.

Probiere die Beispiele in der interaktiven Konsole aus. Die Ergebnisse, die du erhältst werden wahrscheinlich anders aussehen (es handelt sich immerhin um Zufallszahlen).

>>> random.randint(1, 4)

3

>>> random.randint(1000, 2000)

1294

Du kannst den Quellcode und damit das Verhalten des Spiels ändern. Ersetze die Zeilen:

9. zahl = random.randint(1, 20)

10. print('Also, ' + meinName + ', ich denke an eine Zahl zwischen 1 und 20.')

…durch:

9. zahl = random.randint(1, **100**)

10. print('Also, ' + meinName + ', ich denke an eine Zahl zwischen 1 und **100**.')

Jetzt wird der Computer an eine Zahl zwischen 1 und 100, statt an eine zwischen 1 und 20 denken. Durch Ändern von Zeile 9 wird der Wertebereich der Zufallszahlen geändert. Vergiss nicht auch Zeile 10 zu ändern, damit das Spiel dem Spieler den korrekten Wertebereich anzeigt.

Den Spieler begrüßen

10. print('Also, ' + meinName + ', ich denke an eine Zahl zwischen 1 und 20.')

In Zeile 10 heißt die print()-Funktion den Spieler willkommen und teilt ihm mit, dass der Computer an eine Zufallszahl denkt.

Es hat den Anschein, als enthielte Zeile 10 drei Zeichenketten, aber sieh dir die Zeile genau an. Die Pluszeichen verketten die drei Zeichenketten zu einer einzigen. Und das ist das einzige Zeichenketten-Argument, das an die print()-Funktion übergeben wird. Die Kommata befinden sich innerhalb der Anführungszeichen und sind damit Teil der Zeichenkette selbst.

## Schleifen

12. while abgegebenTipps < 6:

Zeile 12 ist eine while-Anweisung, die auf den Beginn einer while-Schleife hindeutet. Schleifen erlauben es dir, einen Codeabschnitt immer wieder auszuführen. Wir müssen jedoch ein paar andere Konzepte durchgehen, bevor wir uns Schleifen genauer ansehen. Dazu gehören Blöcke, Wahrheitswerte, Vergleichsoperatoren, Bedingungen und die while-Anweisung.

## Blöcke

Mehrere Zeilen Programmcode können in einem Block gruppiert werden. Jede Zeile in einem Block hat dieselbe minimale Einrückung. An der Anzahl an Leerzeichen am Anfang der Zeile siehst du, wo ein Block beginnt. Das ist die Einrückung (auf Englisch “indentation”) der Zeile.

Ein Block beginnt wenn die Einrückung der Zeile sich erhöht (normalerweise um vier Leerzeichen). Alle folgenden Zeilen mit gleicher Einrückung sind Teil des Blocks. Der Block endet, sobald eine Zeile mit geringerer Einrückung beginnt. Das heißt, Blöcke können in anderen Blöcken enthalten sein (man sagt auch: Blöcke können ineinander verschachtelt sein). Abbildung 4-1 ist ein Diagramm des Codes, in dem die Blöcke umrissen und durchnummeriert sind.

In Abbildung 4-1 ist Zeile 12 nicht eingerückt und daher auch nicht Teil eines Blocks. Zeile 13 ist vier Leerzeichen eingerückt. Da diese Einrückung größer ist, als die der vorherigen Zeile, beginnt hier ein neuer Block. Der Block ist in Abbildung 4-1 mit (1) gekennzeichnet. Dieser Block erstreckt sich bis zur nächsten Zeile ohne Einrückung. Leerzeilen werden ignoriert.

Zeile 20 hat eine Einrückung von acht Leerzeichen. Acht Leerzeichen sind mehr als vier Leerzeichen, was wiederum einen neuen Block beginnt. Dieser Block ist in Abbildung 4-1 mit (2) gekennzeichnet. Dieser Block ist Teil eines anderen Blocks.

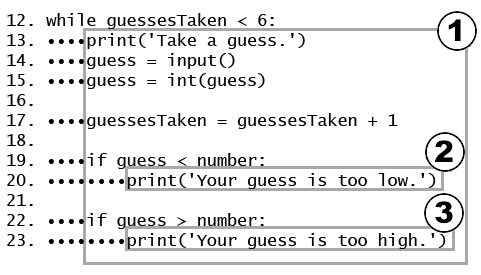


Abbildung 4-1: Blöcke und deren Einrückung. Die schwarzen Quadrate repräsentieren Leerzeichen.

Zeile 22 hat nur vier Einrückungen. Da sich die Einrückung verringert hat, weißt du, dass der Block zu Ende ist. Zeile 20 ist die einzige Zeile in diesem Block. Zeile 22 ist Teil desselben Blocks wie alle anderen Zeilen mit vier Leerzeichen.

Zeile 23 erhöht die Einrückung wieder auf 8 Leerzeichen. Es beginnt also wieder ein neuer Block. Er ist in Abbildung 4-1 mit (3) gekennzeichnet.

Zusammengefasst: Zeile 12 gehört zu keinem Block. Zeilen 13 bis 23 gehören alle zu Block (1). Zeile 20 ist einem Block innerhalb von Block (2). Und Zeile 23 ist die einzige Zeile in einem weiteren Block innerhalb von Block (3).

## Wahrheitswerte

Wahrheitswerte (auch boolesche Werte genannt (auf Englisch “boolean”)) können nur zwei Werte annehmen: True (Englsich für “wahr”) oder False (Englisch für “falsch”). Diese Werte müssen genauso geschrieben werden, wie sie hier aufgeführt sind (mit einem großen “T” bzw. “F” gefolgt von Kleinbuchstaben). Du wirst boolesche Werte mit Vergleichsoperatoren verwenden, um Bedingungen zu formulieren. (Bedingungen werden später erläutert.)

Du kannst Wahrheitswerte natürlich auch in Variablen speichern:

>>> speck = True

>>> eier = False

Die Datentypen, die bis jetzt vorgestellt wurden sind Ganzzahlen (auf Englisch “integer”), Fließkommazahlen (auf Englisch “float”), Zeichenketten (auf Englisch “string”) und nun Wahrheitswerte (auf Englisch “boolean” oder kurz “bool”). Jeder Wert in Python gehört zu einem Datentyp.

## Vergleichsoperatoren

Zeile 12 enthält eine while-Anweisung:

12. while abgegebenTipps < 6:

Der Ausdruck der dem while-Schlüsselwort folgt (der abgegebenTipps < 6-Teil) enthält zwei Werte (der Wert in der Variable abgegebeneTipps und der Ganzzahlenwert 6) verbunden durch einen Operator (das < “kleiner als”-Zeichen). Das <-Zeichen ist ein Vergleichsoperator.

Vergleichsoperatoren vergleichen zwei Werte und werden entweder zu True oder zu False ausgewertet. Eine Liste aller Vergleichsoperatoren findest du in Tabelle 4-1.

Tabelle 4-1: Vergleichsoperatoren.

|  |  |
| --- | --- |
| Operatoren-Symbol | Operatoren-Name |
| < | Kleiner als |
| > | Größer als |
| <= | Kleiner als oder gleich |
| >= | Größer als oder gleich |
| == | Gleich |
| != | Ungleich |

Du kennst schon die mathematischen Operatoren +, -, \* und /. Wie alle Operatoren werden die Vergleichsoperatoren mit Werten kombiniert, um Ausdrücke wie abgegebenTipps < 6 zu bilden.

## Bedingungen

Eine Bedingung ist ein Ausdruck, der zwei Werte mit einem Vergleichsoperator kombiniert (so wie < oder >) und zu einem Wahrheitswert auswertet. Bedingung ist nur ein anderer Name für “Ausdruck der zu True oder False ausgewertet wird”. Bedingungen werden in while-Anweisungen verwendet (und in ein paar anderen Instruktionen, die später erklärt werden).

Die Bedingung abgegebeneTipps < 6 fragt zum Beispiel: “Ist der in abgegebeneTipps gespeicherte Wert kleiner als die Zahl 6?”. Falls ja, dann wir die Bedingung zu True ausgewertet, falls nein, dann wir sie zu False ausgewertet.

In unserem “Rate die Zahl”-Programm wird in Zeile 4 der Wert 0 in abgegebeneTipps gespeichert. Weill 0 weniger als 6 ist, wird diese Bedingung zu True ausgewertet. Die Auswertung würde in etwa so aussehen:

abgegebeneTipps < 6

▼

0 < 6

▼

True

Experimente mit Wahrheitswerten, Vergleichsoperatoren und Bedingungen

Gib die folgenden Ausdrücke in die interaktive Konsole ein und sieh dir an, zu welchen Wahrheitswerten sie ausgewertet werden:

>>> 0 < 6

True

>>> 6 < 0

False

>>> 50 < 10

False

>>> 10 < 11

True

>>> 10 < 10

False

Die Bedingung 0 < 6 wird zu True ausgewertet weil die Zahl 0 kleiner als die Zahl 6 ist. Umgekehrt ist natürlich 6 nicht kleiner als 0 also ergibt 6 < 0 False. 50 ist nicht weniger als 10, also ist 50 < 10 False. 10 ist weniger als 11, also ist 10 < 11 True.

10 < 10 wird zu False ausgewertet weil 10 nicht kleiner als 10 ist, sondern gleich groß. Wenn Alice und Bertl gleich groß sind, dann würdest du auch nicht sagen, dass Alice größer als Bertl sei oder dass Bertl kleiner als Alice sei. Beide Aussagen wären falsch.

Probiere nun folgende Ausdrücke in die interaktive Konsole einzugeben:

>>> 10 == 10

True

>>> 10 == 11

False

>>> 11 == 10

False

>>> 10 != 10

False

>>> 10 != 11

True

>>> 'Hallo' == 'Hallo'

True

>>> 'Hallo' == 'Tschüss'

False

>>> 'Hallo' == 'HALLO'

False

>>> 'Tschüss' != 'Hallo'

True

## Der Unterschied zwischen = und ==

Der Zuweisungsoperator (=) und der Gleichheitsoperator (==) sind leicht zu verwechseln. Das einfache Gleichheitszeichen (=) wird in Zuweisungsausdrücken verwendet, um einen Wert in einer Variable zu speichern, das doppelte Gleichheitszeichen (==) wird verwendet, um zu testen, ob zwei Werte gleich sind.

Es gibt eine Eselsbrücke, um die beiden Operatoren auseinander zu halten: Der “ist gleich”-Operator (==), sowie der “ist nicht gleich”-Operator (!=) bestehen beide aus zwei Zeichen.

Zeichenketten und Ganzzahlen können niemals gleich zueinander sein. Probiere Folgendes in die interaktive Konsole einzugeben:

>>> 42 == 'Hallo'

False

>>> 42 != '42'

True

## Iterieren mit while-Anweisungen

Die while-Anweisung (Englisch für “solange”) markiert den Beginn einer Schleife. Schleifen führen denselben Code wiederholt aus (was man auch als “iterieren” bezeichnet). Wenn die Programmausführung die while-Anweisung erreicht, dann wird die Bedingung neben dem while-Schlüsselwort ausgewertet. Ist die Bedingung wahr, so begibt sich die Ausführung in den folgenden Block, while-Block genannt. (In obigem Programm beginnt der while-Block in Zeile 13.) Ist die Bedingung falsch, dann wird der while-Block übersprungen und die Ausführung in der ersten Zeile nach dem while-Block fortgesetzt. In Rate die Zahl ist Zeile 28 die erste Zeile nach dem while-Block.

Eine while-Anweisung enthält immer einen Doppelpunkt : nach der Bedingung. Anweisungen, die mit einem Doppelpunkt enden erwarten, dass in der nächsten Zeile ein neuer Block beginnt.

12. while abgegebenTipps < 6:

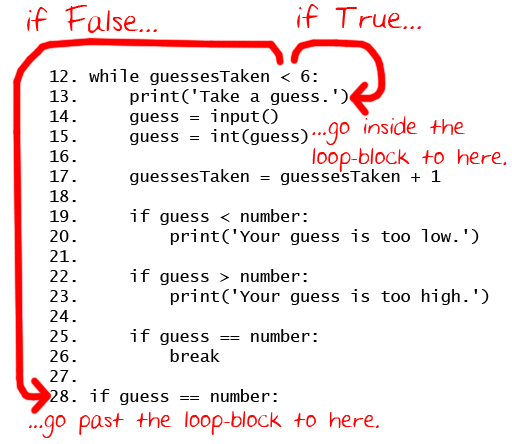


Abbildung 4-2: Die Bedingung der while-Schleife

Abbildung 4-2 zeigt den Verlauf der Programmausführung in Abhängigkeit von der Bedingung. Wenn die Bedingung zu True ausgewertet wird (wie es beim ersten Durchlauf des Programms der Falls ist, weil der Wert von abgegebeneTipps 0 ist), dann wird nach der while-Anweisung der while-Block beginnend in Zeile 13 ausgeführt. Wenn das Programm das Ende des while Blocks erreicht, dann wird es nicht in der Zeile darunter fortfahren, sondern wieder zurück zur while-Anweisung (Zeile 12) springen und deren Bedingung auswerten (man sagt auch “evaluieren”). Ist die Bedingung True, betritt die Ausführung wie zuvor in den while-Block. Jeder dieser Schleifendurchgänge wird als Iteration bezeichnet.

Die while-Schleife funktionier folgendermaßen: Solange die Bedingung wahr ist, führt das Programm immer wieder den while-Block aus, bis die Bedingung zum ersten Mal False ist. Die while-Anweisung könnte man in Worten so ausdrücken: “solange (while) diese Bedingung wahr (True) ist, führe den Code im folgenden Block aus”.

Der Spieler rät

13. print('Los, rate.') # Vor print sind vier Leerzeichen.

14. tipp = input()

Zeilen 13 bis 17 fordern den Spieler auf, eine Zahl zu raten und diese einzugeben. Diese Zahl wird in der Variable tipp gespeichert.

## Zahlen mit den Funktionen int(), float() und str() umwandeln

15. tipp = int(tipp)

Zeile 15 ruft eine neue Funktion mit dem Namen int() auf. Die int()-Funktion erwartet ein Argument und gibt den Ganzzahlenwert des Arguments zurück. Probiere Folgendes in die interaktive Konsole einzugeben:

>>> int('42')

42

>>> 3 + int('2')

5

Der Funktionsaufruf int('42') gibt den Ganzzahlenwert 42 zurück. int() kann allerdings nicht beliebige Zeichenketten in Ganzzahlen umwandeln. Wenn z.B. 'zweiundvierzig' übergeben würde, würde das in einem Fehler resultieren. Die an int() übergebene Zeichenkette muss aus Zahlen bestehen:

>>> int('zweiundvierzig')

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

int('zweiundvierzig')

ValueError: invalid literal for int() with base 10: 'zweiundvierzig'

The 3 + int('2') line shows an expression that uses the return value of int() as part of an expression. It evaluates to the integer value 5:

3 + int('2')

▼

3 + 2

▼

5

Remember, the input() function always returns **a string** of text the player typed. If the player types 5, the input() function will return the string value '5', not the integer value 5. Python cannot use the < and > comparison operators to compare a string and an integer value:

>>> 4 < '5'

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

4 < '5'

TypeError: unorderable types: int() < str()

14. tipp = input()

15. tipp = int(tipp)

On line 14 the guess variable originally held the string value of what the player typed. Line 15 overwrites the string value in guess with the integer value returned by int(). This lets the code later in the program compare if guess is greater than, less than, or equal to the secret number in the number variable.

One last thing: Calling int(guess) doesn’t change the value in the guess variable. The code int(guess) is an expression that evaluates to the integer value form of the string stored in the guess variable. What changes guess is the assignment statement: guess = int(guess)

The float(), str(), and bool() functions will similarly return float, string, and Boolean versions of the arguments passed to them. Try entering the following into the interactive shell:

>>> float('42')

42.0

>>> float(42)

42.0

>>> str(42)

'42'

>>> str(42.0)

'42.0'

>>> str(False)

'False'

>>> bool('')

False

>>> bool('any nonempty string')

True

Using the int(), float(), str(), and bool() functions, you can take a value of one data type and return it as a value of a different data type.

Incrementing Variables

17. abgegebenTipps = abgegebenTipps + 1

Once the player has taken a guess, the number of guesses should be increased by one.

On the first iteration of the loop, guessesTaken has the value of 0. Python will take this value and add 1 to it. 0 + 1 evaluates to 1, which is stored as the new value of guessesTaken. Think of line 17 as meaning, “the guessesTaken variable should be one more than what it already is”.

Adding one to a variable’s integer or float value is called incrementing the variable. Subtracting one from a variable’s integer or float value is called decrementing the variable.

## if statements

19. if tipp < zahl:

20. print('Dein Tipp ist zu niedrig.') # Hier sind acht Leerzeichen vor print.

Line 19 is an if statement. The execution will run the code in the following block if the if statement’s condition evaluates to True. If the condition is False, then the code in the if-block is skipped. Using if statements, you can make the program only run certain code when you want it to.

Line 19 checks if the player’s guess is less than the computer’s secret number. If so, then the execution moves inside the if-block on line 20 and prints a message telling the player this.

The if statement works almost the same as a while statement, too. But unlike the while-block, the execution doesn’t jump back to the if statement at the end of the if-block. It just continues down to the next line. In other words, if statements don’t loop. See Figure 4-3 for a comparison of the two statements.

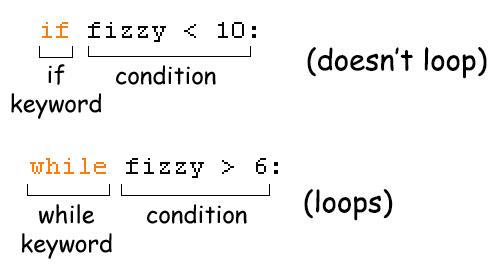


Figure 4-3: if and while statements.

22. if tipp > zahl:

23. print('Dein Tipp ist zu hoch.')

Line 22 checks if the player’s guess is greater than the secret number. If this condition is True, then the print() function call tells the player that their guess is too high.

## Leaving Loops Early with the break statement

25. if tipp == zahl:

26. break

The if statement on line 25 checks if the guess is equal to the secret number. If it is, the program runs the break statement on line 26.

A break statement tells the execution to jump immediately out of the while-block to the first line after the end of the while-block. The break statement doesn’t bother rechecking the while loop’s condition.

The break statement is only found inside loops, such as in a while-block.

If the player’s guess isn’t equal to the secret number, the execution reaches the bottom of the while-block. This means the execution will loop back to the top and recheck the condition on line 12 (guessesTaken < 6). Remember after the guessesTaken = guessesTaken + 1 instruction executed, the new value of guessesTaken is 1. Because 1 < 6 is True, the execution enters the loop again.

If the player keeps guessing too low or too high, the value of guessesTaken will change to 2, then 3, then 4, then 5, then 6. When guessesTaken has the number 6 stored in it, the while statement’s condition (guessesTaken < 6) is False, since 6 isn’t less than 6. Because the while statement’s condition is False, the execution moves to the first line after the while-block, line 28.

Check if the Player Won

28. if tipp == zahl:

Line 28 has no indentation, which means the while-block has ended and this is the first line after the while-block. The execution left the while-block either because the while statement’s condition was False (when the player runs out of guesses) or the break statement on line 26 was executed (when the player guesses the number correctly).

Line 28 checks to see if the player guessed correctly. If so, the execution enters the if-block at line 29.

29. abgegebenTipps = str(abgegebenTipps)

30. print('Gut gemacht, ' + meinName + '! Du hast meine Zahl in ' + abgegebenTipps + ' Zügen erraten!')

Lines 29 and 30 only execute if the condition in the if statement on line 28 was True (that is, if the player correctly guessed the computer’s number).

Line 29 calls the str() function, which returns the string form of guessesTaken. Line 30 concatenates strings to tell the player they have won and how many guesses it took them. Only string values can concatenate to other strings. This is why line 29 had to change guessesTaken to the string form. Otherwise, trying to concatenate a string to an integer would cause Python to display an error.

Check if the Player Lost

32. if tipp != zahl:

Line 32 uses the “not equal to” comparison operator != to check if player’s last guess is not equal to the secret number. If this condition evaluates to True, the execution moves into the if-block on line 33.

Lines 33 and 34 are inside the if-block, and only execute if the condition on line 32 was True.

33. zahl = str(zahl)

34. print('Nene. Die Nummer an die ich gedacht habe war ' + zahl)

In this block, the program tells the player what the secret number they failed to guess correctly was. This requires concatenating strings, but number stores an integer value. Line 33 will overwrite number with a string form so that it can be concatenated to the 'Nope. The number I was thinking of was ' string on line 34.

At this point, the execution has reached the end of the code, and the program terminates. Congratulations! You’ve just programmed your first real game!

You can change the game’s difficulty by changing the number of guesses the player gets. To give the player only four guesses, change the code on line 12:

12. while abgegebenTipps < 6:

into this line:

12. while abgegebenTipps < 4:

Code later in the while-block increases the guessesTaken variable by 1 on each iteration. By setting the condition to guessesTaken < 4, you ensure that the code inside the loop only runs four times instead of six. This makes the game much more difficult. To make the game easier, set the condition to guessesTaken < 8 or guessesTaken < 10. This will cause the loop to run a few more times and accept more guesses from the player.

## Flow Control Statements

In previous chapters, the program execution started at the top instruction in program and went straight down, executing each instruction in order. But with the while, if, else, and break statements, you can cause the execution to loop and skip instructions based on conditions. The name for these kinds of statements is flow control statement, since they change the “flow” of the program execution as it moves around your program.

Summary

If someone asked you, **“What exactly is programming anyway?”** what could you say to them? Programming is just the action of writing code for programs, that is, creating programs that can be executed by a computer.

**“But what exactly is a program?”** When you see someone using a computer program (for example, playing your “Guess the Number” game), all you see is some text appearing on the screen. The program decides what exact text to show on the screen (the program’s output), based on its instructions and on the text that the player typed on the keyboard (the program’s input). A program is just a collection of instructions that act on the user’s input.

**“What kind of instructions?”** There are only a few different kinds of instructions, really.

1. **Expressions** are values connected by operators. Expressions are all evaluated down to a single value, as 2 + 2 evaluates to 4 or 'Hello' + ' ' + 'World' evaluates to 'Hello World'. When expressions are next to the if and while keywords, you can also call them conditions.
2. **Assignment statements** store values in variables so you can remember the values later in the program.
3. **The** if**,** while**, and** break **statements** are flow control statements that can cause the execution to skip instructions, loop over instructions, or break out of loops. Function calls also change the flow of execution by jumping to the instructions inside of a function.
4. **The** print() **and** input() **functions.** These functions display text on the screen and get text from the keyboard. This is called I/O (pronounced like the letters, “eye-oh”), because it deals with the Input and Output of the program.

And that’s it, just those four things. Of course, there are many details about those four types of instructions. In this book you’ll learn about new data types and operators, new flow control statements, and many other functions that come with Python. There are also different types of I/O such as input from the mouse or outputting sound and graphics instead of just text.

For the person using your programs, they only care about that last type, I/O. The user types on the keyboard and then sees things on the screen or hears things from the speakers. But for the computer to figure out what sights to show and what sounds to play, it needs a program, and programs are just a bunch of instructions that you, the programmer, have written.