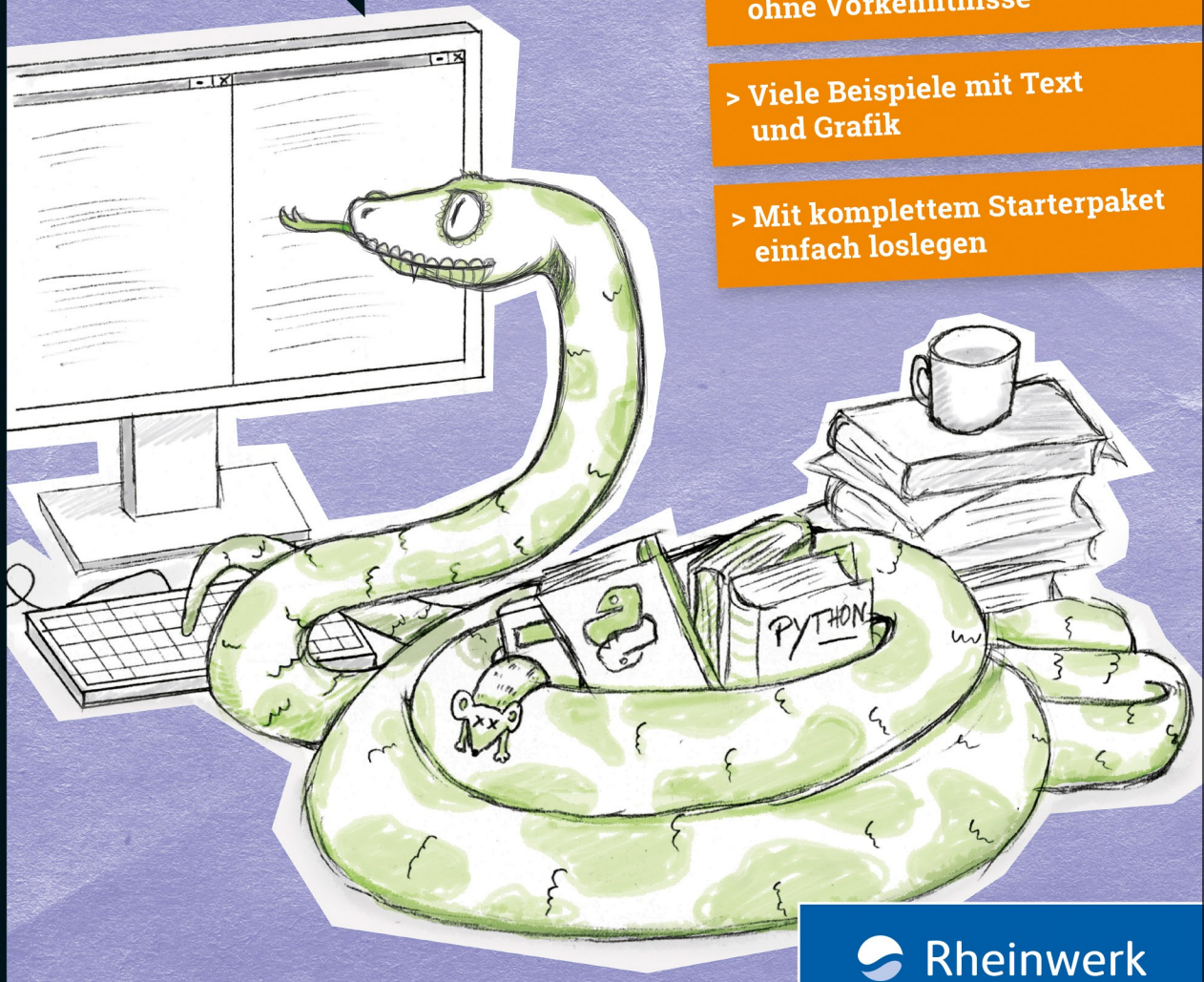
Hauke Fehr

<LET'S CODE> Python

> Programmieren lernen ohne Vorkenntnisse

> Viele Beispiele mit Text und Grafik

> Mit komplettem Starterpaket einfach loslegen



2., aktualisierte Auflage

Rheinwerk Computing

Kapitel 1

Programme schreiben – wie geht das?

Wer selbst Computerprogramme schreiben kann, ist ein Programmierer. Und programmieren lernen kann jede und jeder – dafür braucht man kein Informatikstudium. Wer einmal verstanden hat, was Programmieren eigentlich ist, und es Schritt für Schritt ausprobiert, der kann bald programmieren wie ein Profi, versprochen!

Du möchtest Programmierer werden? Herzlichen Glückwunsch, das ist eine gute Entscheidung. Programmieren kann ungeheuren Spaß machen. Es ist eine Tätigkeit, die dein Gehirn auf allen Ebenen fordert. Programmieren fördert gleichzeitig logisches und kreatives Denken. Du brauchst beides, um gute Programme zu schreiben – aber keine Angst. Es ist nicht erforderlich, dass du ein Mathegenie bist oder eine geniale Erfinderin. Wenn du Spaß am Programmieren hast, findest du deinen eigenen Bereich, in dem du alle deine Fähigkeiten nutzen kannst.

Was bedeutet denn Programmieren eigentlich? Ist es so ähnlich wie das Schreiben von Texten in Word oder das Erstellen einer PowerPoint-Präsentation?

Ja und nein. Software zu schreiben ist etwas anderes als Software zu verwenden. Es ist wie mit Autos – viele können sie fahren, wenige können sie reparieren, und noch weniger Menschen können eigene Autos entwerfen und bauen. So ist es mit vielen Programmen auch. Viele können zum Beispiel gut mit Office-Programmen umgehen oder auch Bilder mit Grafikeditoren bearbeiten. Und insbesondere Spiele beherrschen viele am Computer in großer Meisterschaft.

Eigene Programme zu entwickeln ist aber noch einmal etwas ganz anderes. Hier bist du von der Idee bis zur perfekten fertigen App ganz allein der Regisseur. Du denkst dir aus, was dein Programm können soll – und dann setzt du es Schritt für Schritt um, ganz nach deinen Ideen und deinem Können. Mit jedem Programm, das du schreibst, erweiterst du deine Fähigkeiten und kannst dich danach an ein noch größeres oder spannenderes Projekt wagen. Immerhin kostet Programmieren nur Zeit, kein Geld, solange dir ein Computer zur Verfügung steht. Du brauchst keine Autowerkstatt oder Produktionsanlage, sondern nur deine Fantasie und das Wissen, wie man herangeht.

Sollte man dafür nicht besser Informatik studieren?

Nein, das brauchst du nicht, um Programmieren zu lernen. Wenn du irgendwann im IT-Bereich einer Firma einen guten Job haben willst, ist es sicher von Vorteil, ein Informatikstudium vorzeigen zu können. Aber das Programmieren kannst du vollkommen selber lernen. Mit den vielen, vielen Anleitungen und Tutorial-Videos im Internet ist es heutzutage besonders einfach geworden, sich jedes Spezialgebiet des Programmierens anzueignen. Je mehr Erfahrung du hast, desto schneller lernst du neue Techniken, denn alles baut aufeinander auf. Ich selbst habe übrigens nie Informatik studiert und lebe trotzdem seit 25 Jahren davon, Software für alle Bereiche zu programmieren und zu verkaufen.

Was muss ich denn können, um Programmieren zu lernen?

Die besten und wichtigsten Voraussetzungen sind Neugier, Spaß und Lust darauf. Wenn du gerne am Computer arbeitest, dir auch mal ab und zu etwas Eigenes ausdenkst, wenn du neugierig bist, wie man ein Problem löst oder wie Dinge am Computer hinter den Kulissen gemacht werden, wenn du Spaß an einfachen oder auch kniffligeren Aufgaben hast oder auch nur Freude am Gestalten, Steuern und Basteln, dann bist du genau auf dem richtigen Kurs, selbst Programmierer zu werden.

Du brauchst kein Mathegenie zu sein – um Mathe geht es nur am Rande –, da reicht die Fähigkeit, dir Zahlen und Größen ein wenig vorstellen zu können. Du brauchst auch kein großartiger Gestalter zu sein – wenn doch, dann kannst du das in deine Programme mit einbringen, wenn nein, dann findest du einen anderen Bereich der Programmierung, der dir naheliegt.

Das Programmieren ist sehr vielseitig, und jeder Bereich, der dich fasziniert, ist erlernbar. Man kann mit Programmen Alltagsaufgaben lösen, knifflige Probleme in den Griff bekommen, kreative Spiele bauen, Roboter steuern, Lernmedien erstellen, Daten verwalten, alles, was ein Computer kann, kannst du prinzipiell auch selber programmieren.

Wie und womit fange ich am besten das Programmieren an und wie weit kann ich kommen?

Nun, die erste Frage ist einfach zu beantworten: Du fängst am besten mit diesem Buch an. Python zu lernen ist ein hervorragender Einstieg in das Programmieren. Python ist einfacher zu lernen als viele andere Sprachen, und trotzdem ist das, was du mit Python als Profi machen kannst, am Ende unbegrenzt. Mithilfe von Modulen aus allen nur erdenklichen Bereichen steht dir die ganze Welt des Programmierens offen, sei es die Programmierung von Datenbanken, Webservern, von Spielen, praktischen Tools oder Steuerungen – alles ist vollständig in Python möglich.

Python enthält alle Befehle und Strukturen, die andere professionelle Sprachen auch haben, bis hin zu den komplexesten Methoden. Wenn du in Python verstanden hast, wie Programmiersprachen funktionieren, kannst du später bei Bedarf auch leicht umsteigen auf andere Programmiersprachen wie Java, C++, JavaScript, PHP oder was immer du beruflich oder privat nutzen willst. Mit Python kannst du alle wichtigen Grundlagen und Verfahren erlernen, die alle Programmierer in jedem System immer wieder anwenden.

Welche Version von Python lerne ich in diesem Buch?

In diesem Buch lernst du die grundlegenden Programmiertechniken mit *TigerJython*, einer ebenso einfachen wie mächtigen Lern- und Entwicklungsumgebung für Python, mit der du entdecken, programmieren, testen und spielen kannst. Die Befehle und Strukturen, die du hier lernst, funktionieren sowohl in Python 2 als auch im neueren Python 3.

Wie schnell lerne ich das Programmieren?

Das hängt natürlich ganz von dir selbst ab, davon, wie viel Zeit und Energie du in das Lernen hineinsteckst, wie viel Spaß es dir macht, dich immer wieder neu herauszufordern. Du musst am Anfang erst einmal wie ein Programmierer denken lernen. Das braucht ein bisschen Zeit, aber es geht dann nach und nach immer schneller und immer leichter. Am Anfang sind es ganz kleine Programme, die du schreiben wirst, aber mit jedem Programm, das du selbst schreibst oder anpasst, lernst du etwas dazu. Dieses Buch will dir dazu alle wichtigen Grundlagen vermitteln – das Wissen, wie man als Programmierer an Aufgaben herangeht, wie man ein Programm plant und aufbaut, welche Strukturen man wofür verwendet, welche Grundbefehle und erweiterten Module es gibt, wie man typische Vorgänge in ein Programm umsetzt. Vom einfachsten »Hallo« auf dem Bildschirm bis hin zur objektorientierten Spieleprogrammierung deckt dieses Buch sehr viel ab – natürlich immer mit einfachen und direkt umsetzbaren Beispielen. Wenn du das Buch durchgearbeitet und alle Beispiele wirklich selbst ausprobiert und verstanden hast, dann darfst du dich Programmierer oder Programmiererin nennen, und du kannst dich dann selbst an größere eigene Projekte machen. Von da an steht dir die Welt offen. Wohin deine Reise als Programmierer weitergeht, bestimmst du ganz allein – das Wissen und die Fähigkeit zu programmieren hast du dann bereits!

Kapitel 7

Programme schreiben – es geht los!

Jetzt haben wir genug Zeit mit dem Testen auf der Konsole verbracht. Ein Programmierer will programmieren. Und dafür haben wir jetzt die wichtigsten Grundlagen beisammen. Es kann also beginnen.

Du hast bislang Python einzelne Befehle gegeben. Aber Befehle sind noch keine Programme. Ein Programm setzt sich zwar aus Befehlen zusammen, aber erst, wenn du mehrere Befehle zu einer Liste von Anweisungen verknüpfst, hast du ein Programm.

Fassen wir kurz zusammen: Welche Anweisungen kennst du?

- die `input()`-Funktion, mit der du die Eingabe des Benutzers in eine Variable schreiben kannst
- die Variablenzuweisung, mit der du eine Zahl oder eine Zeichenkette in eine Variable schreiben kannst
- die Berechnung von Werten mit mathematischen Operatoren oder die Verknüpfung von Zeichenketten
- die `print`-Funktion, um einen Wert oder den Wert einer Variablen im Ausgabebereich auszugeben, und die `msgDlg()`-Funktion, um einen Wert in einem Aufklappfenster anzuzeigen

Das sind im Grunde nur vier Arten von Anweisungen, aber mit denen kannst du schon eine Menge anfangen. Denn mit ihnen kannst du die wichtigsten Grundbestandteile eines Computerprogramms erzeugen: Eingabe von Daten, Verarbeitung von Daten, Ausgabe des Ergebnisses.

Ein Programm in TigerJython eingeben

Die Konsole kannst du jetzt schließen. Du wirst sie später noch hin und wieder zum Testen verwenden, aber dein Arbeitsbereich ist von nun an das Hauptfenster von TigerJython.

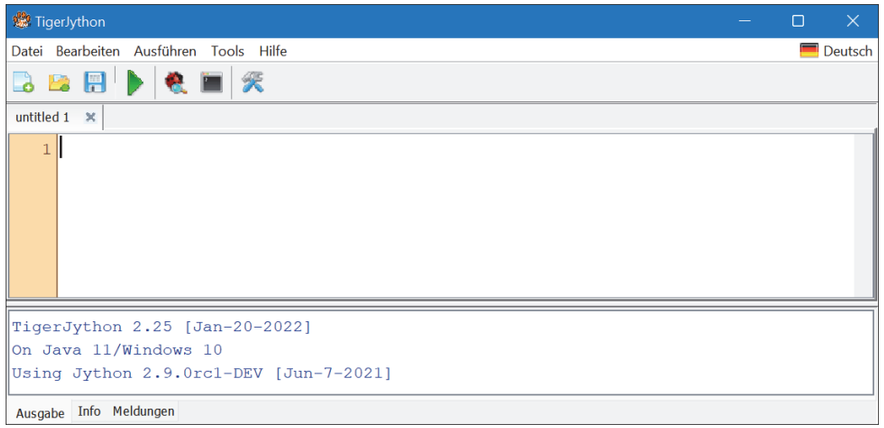


Abbildung 7.1 Ab jetzt wird im Programmierfenster gearbeitet.

In die große weiße Fläche wirst du von jetzt an die einzelnen Anweisungen des Programms nacheinander eingeben. Mit der ↵-Taste gelangst du in die nächste Zeile, aber der Befehl wird dabei nicht wie in der Konsole direkt ausgeführt.

Erst wenn dein Programm fertig ist oder du es testen möchtest, startest du das Programm mit einem Klick auf das Symbol mit dem grünen Dreieck ( ).

Damit werden die Befehle dann von oben nach unten Zeile für Zeile nacheinander abgearbeitet. Das bedeutet es, *ein Programm auszuführen*.

Das allererste Programm: Ein Zahlenzaubertrick

Zum Warmwerden ein allererstes Programm, das aus einer Abfolge von Ausgaben besteht. Hier werden noch keine Variablen verwendet, keine Eingabe von Werten, nur die Ausgabe von Texten. Das kann man zum Beispiel mit aneinandergereihten `msgDlg()`-Befehlen machen (Text als Nachricht im Fenster ausgeben), die einfach nur ihren Text zeigen und dann nach dem Klick auf OK wieder verschwinden.

Gib mal folgendes Programm in das Programmfenster ein:

```
msgDlg("Denke dir eine beliebige Zahl zwischen eins und zehn aus!")
msgDlg("Multipliziere die Zahl mit 5.")
msgDlg("Verdopple die Zahl.")
msgDlg("Teile die Zahl jetzt durch die Zahl, die du dir am Anfang ausgedacht hast.")
msgDlg("Ziehe 7 von der momentanen Zahl ab.")
```



```
msgDlg("Jetzt sage ich dir, welche Zahl du gerade hast. Es ist die ...")
msgDlg("DREI!")
```

Programme aus dem Buch eingeben

Ich würde dir empfehlen, die Programme aus dem Buch, insbesondere am Anfang, immer selbst abzuschreiben. Dadurch entwickelst du allmählich ein Gefühl dafür, wie man die Befehle und Strukturen richtig verwendet. Doch für den Fall, dass du ein Programm schnell testen willst oder den Fehler in deinem abgeschriebenen Programm nicht finden kannst, gibt es alle Programme aus dem Buch auch als HTML-Datei in den »Materialien zum Buch«. Du kannst sie von dort ganz einfach durch Anklicken und Einfügen in TigerJython direkt übernehmen.

Du findest die Materialien hier:

www.rheinwerk-verlag.de/5399/

Dort wählst du MATERIALIEN.

Außerdem gibt es die aktuellste Fassung der Skripte und weitere Hinweise, Links, eventuelle Korrekturen und Erläuterungen auf meiner Website zum Buch:

www.letscode-python.de

Der `msgDlg`-Befehl in TigerJython ist nur für kurze Mitteilungen an den Benutzer gedacht. Hier wird er verwendet, um immer wieder einen Text auszugeben und abzuwarten, bis der Benutzer auf OK klickt oder die ↵-Taste drückt, damit es weitergehen kann.

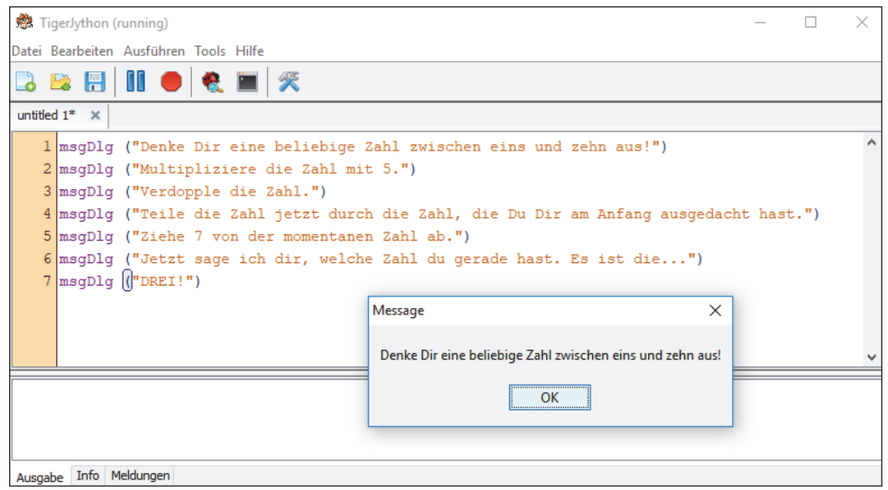


Abbildung 7.2 Das Programm gibt seine Nachrichten nacheinander in Dialogboxen aus.

Probiere es aus! Klicke auf den grünen Pfeil, und das Programm startet. Denk dir eine Zahl aus und folge weiter den Anweisungen. Du musst nichts eingeben, sondern nur auf OK klicken.

Wenn du richtig rechnest, klappt der Zaubertrick, und »DREI!« am Ende stimmt. Zugegeben, der Trick ist leicht zu durchschauen, aber viel wichtiger ist: Du hast ein erstes Python-Programm geschrieben, das funktioniert!

Python führt die Anweisungen Schritt für Schritt aus, beginnend mit der obersten Zeile und dann immer eine nach der anderen. Der `msgDlg`-Befehl ist immer erst beendet, wenn der Benutzer auf OK klickt – dann wird der nächste Befehl ausgeführt. So entsteht etwas wie ein Gespräch mit dem Programm, das dir nach und nach Mitteilungen macht.

Nach dieser Aufwärmübung soll nun das erste »richtige« Programm folgen, das alle drei wichtigen Grundelemente besitzt: *Eingabe, Verarbeitung, Ausgabe.*

Zweites Programm: Ein Umrechner

Die Aufgabe ist folgende: Du möchtest ein Programm schreiben, das eine eingegebene Längenangabe in Zoll (wie zum Beispiel eine Bildschirmgröße) in Zentimeter umwandelt und das Ergebnis ausgibt.

Welche Schritte muss so ein Programm ausführen?

- Schritt 1: Eingabe der Länge in Zoll, Speichern in einer Variablen, zum Beispiel mit dem Namen `laenge_zoll`
- Schritt 2: Berechnen der Länge in cm und Speichern des Ergebnisses in einer Variablen, zum Beispiel mit dem Namen `laenge_cm`
  (1 Zoll = 2,54 cm)
- Schritt 3: Ausgabe des Ergebnisses (`laenge_cm`) mit erläuterndem Text

Jeder dieser Schritte muss natürlich in Python richtig formuliert werden. Kannst du das allein?

Ansonsten findest du hier eine mögliche Lösung:

```
laenge_zoll = input("Bitte Länge in Zoll eingeben:")
laenge_cm = laenge_zoll * 2.54
print("Ergebnis:", laenge_cm , "Zentimeter")
```

Nur drei Zeilen genügen. In der ersten wird mit dem `input`-Befehl eine Zahl eingegeben und in der Variablen `zoll` gespeichert.

In der zweiten wird `zoll * 2,54` berechnet (das ist die Umrechnungsformel von Zoll in Zentimeter) und in der Variablen `cm` gespeichert.

In der dritten Zeile wird ausgegeben: `Ergebnis:`, dann der berechnete Wert in der Variablen `cm`, dann `Zentimeter`.

Probiere es aus: Klicke auf das grüne Dreieck, und das Programm läuft los.

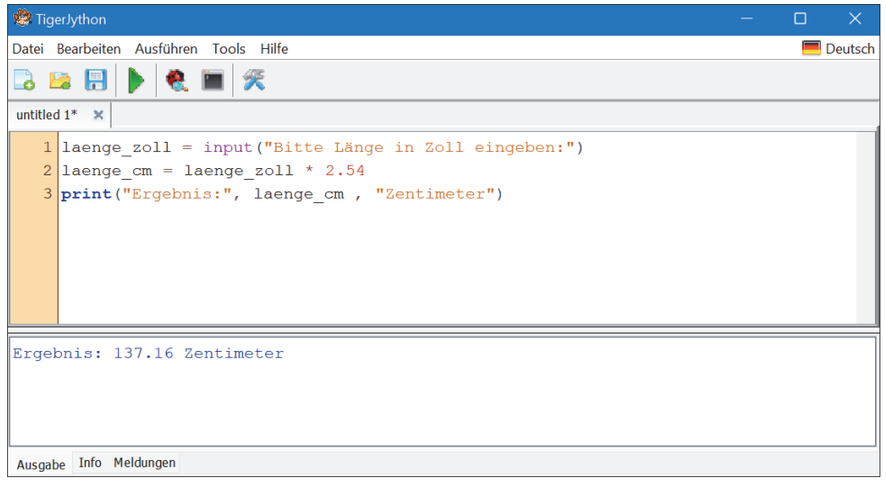


Abbildung 7.3 Ergebnis in Zentimeter: hier 137,16 cm

Funktioniert es?

Herzlichen Glückwunsch! Du hast dein erstes klassisches Programm geschrieben!

Und wo du gerade so schön dabei bist – wie wäre es, wenn du das Programm abänderst, sodass es zum Beispiel Meter in Fuß umrechnet (1 Meter = 3,2808 Fuß)? Oder Geschwindigkeit: Knoten in km/h (1 Knoten = 1,852 km/h)? Oder Stunden in Sekunden (1 Stunde hat 3.600 Sekunden)? Oder deine ganz eigene Idee?

Aufgabe

Verändere das Programm so, dass es andere von dir festgelegte Einheiten umrechnen kann. Passe die Berechnung und den Text an (und am besten auch die Namen der Variablen). Teste es und prüfe, ob es funktioniert.

Programme speichern

Ein funktionierendes Programm möchte man natürlich auch behalten. Deshalb solltest du jedes Programm, das du eingegeben hast, wenn es fertig ist (oder auch mal zwischendurch) speichern. Dann kann es dir nicht mehr verloren gehen, und du kannst es jederzeit wieder aufrufen, verwenden oder verändern.

Leg dir einen Python-Ordner an!

Ich würde dir empfehlen, an dieser Stelle einen Ordner auf deiner Festplatte anzulegen, in dem du zukünftig alle deine Python-Programme speicherst. Damit schaffst du Ordnung und weißt immer, wo du Python-Programme suchen musst. Der Ordner kann zum Beispiel *Pythonprogramme* heißen und im Verzeichnis *Dokumente* erstellt werden. Wichtig ist, dass du ihn in einem deiner persönlichen Verzeichnisse anlegst, damit du auch vollen Zugriff darauf hast.

Das Speichern selbst ist sehr einfach. Wähle im Menü von TigerJython einfach Datei • Speichern unter...

Nun musst du dein selbst angelegtes Python-Verzeichnis wählen, deinem Programm einen kurzen, aber treffenden Namen geben und auf Speichern klicken.

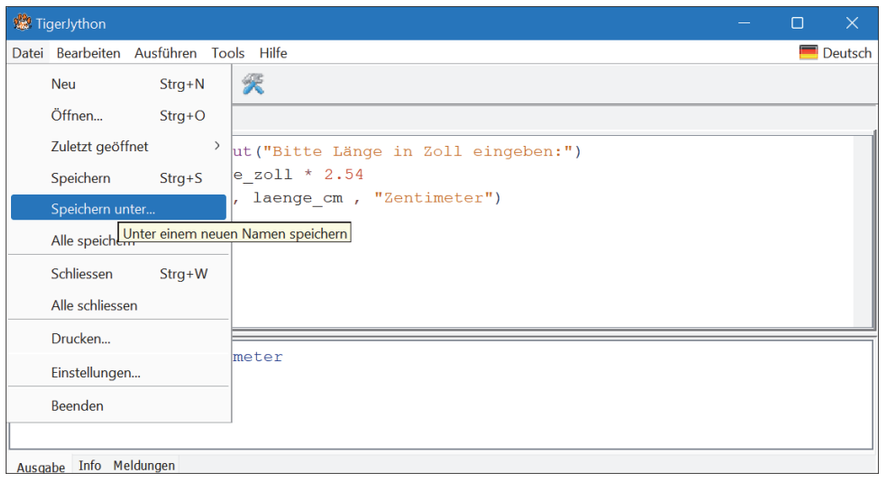


Abbildung 7.4 Mit »Speichern unter...« kannst du ein Programm erstmalig abspeichern.

Wenn du das Programm später noch bearbeitest und dann erneut speichern willst, reicht es aus, Strg + S (bzw. auf dem Mac cmd + S) zu drücken, und das Programm wird unter seinem vorhandenen Namen erneut gespeichert.

Laden kannst du gespeicherte Programme jederzeit, indem du Datei • Öffnen im Menü wählst (oder Strg + O bzw. cmd + O), das gewünschte Python-Programm auswählst und auf Öffnen klickst.

Eingabe, Verarbeitung, Ausgabe – diesmal mit Text

Wie du ja weißt, kann Python nicht nur mit Zahlen umgehen, sondern auch mit Texten (Zeichenketten bzw. Strings). Das probieren wir gleich mal im nächsten Beispiel aus, indem wir Texte zu einer Begrüßung verknüpfen. Auf der Konsole haben wir es schon getestet, jetzt wird ein richtiges kleines Programm mit nur zwei Zeilen daraus:

```
name = input("Wie heißt du?")
print("Freut mich, dich kennenzulernen, " + name + "!")
```

Bevor du es startest: Kannst du schon aus dem Code ersehen, was es machen wird? Wahrscheinlich schon, oder? Probiere es aus, indem du den grünen Pfeil klickst und testest, ob deine Vermutung stimmt.

Noch ein Beispiel, das diesmal die »Multiplikation von Text« verwendet:

```
name = input("Wie heißt du?")
anzahl = input("Wie oft soll ich dich begrüßen?")
print(("Hallo "+name+"! ")*anzahl)
```

Kannst du ermitteln, was dieses Programm genau machen wird?

Zuerst wird wieder dein Name eingegeben und in der Variablen `name` gespeichert. Dann wird eine Anzahl eingegeben und in der Variablen `anzahl` gespeichert.

Und zum Schluss wird »Hallo« mit dem eingegebenen Namen und mit einem Ausrufezeichen und Leerzeichen verknüpft. Das Ganze (daher ist es eingeklammert) soll multipliziert werden, also so viele Male, wie in `anzahl` steht, nacheinander erscheinen.

Teste es!

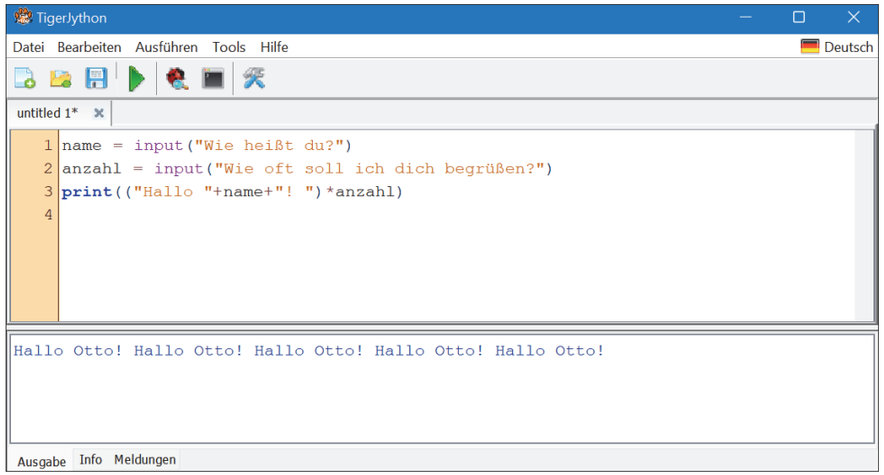


Abbildung 7.5 Fünfmal nacheinander wird hier Otto begrüßt.

Du kannst als Anzahl gern auch mal 100 oder gar 1.000 eingeben. Dann erhältst du eine sehr lange Begrüßungsreihe im Ausgabefenster.

Übrigens: Wenn du möchtest, dass dein Name nicht nebeneinander, sondern untereinander ausgegeben wird, musst du die Ausgabezeile so verändern:

```
print(("Hallo "+name+"!"+"\n")*anzahl)
```

An den Namen wird also noch ein `\n` angehängt – `\n` ist ein Steuerzeichen und steht für *new line*, also »neue Zeile«.

Mit dem, was du jetzt weißt, kannst du dir auch schon eigene kleine Programme ausdenken. Überleg dir etwas, und probiere es aus. Durch Probieren lernt man immer am meisten!

Rechner mit Rest

Noch ein kleines Beispiel, diesmal für einen Rechner. Ein normaler Taschenrechner kann zwar alles Mögliche, aber meistens kann er keine Division mit Rest ausführen, sondern er rechnet automatisch in Dezimalbrüche um. Mit Python kann man, wie wir schon auf der Konsole gesehen haben, sehr einfach das Ganzzahlergebnis und den Rest einer Geteilt-Aufgabe ermitteln. Machen wir doch auch ein kleines Programm daraus.

Was muss das Programm leisten?

- Eingabe der Grundzahl (g)
- Eingabe des Teilers (t)
- Berechnung des Ganzzahlergebnisses (e)
- Berechnung des Restes (r)
- Ausgabe von Ergebnis und Rest

Weißt du noch, wie das in Python geht? Wenn du in Python sicher werden willst, dann empfehle ich dir zu versuchen, die Programme immer erst einmal selbst zu schreiben, bevor du den Lösungsvorschlag im Buch anschaust und abtippst. Wie die Division mit Ganzzahl und Rest genau funktioniert, kannst du in Kapitel 5, »Die ersten Schritte – Python im Dialog«, nachschlagen.

Und hier der Vorschlag, wie man ein solches Programm schreiben kann:

```
g = input("Wie lautet die Grundzahl?")
t = input("Durch welche Zahl soll sie geteilt werden?")
e = g // t
r = g % t
print("Das Ergebnis ist", e ,"und der Rest ist", r)
```

Zuerst wird die Grundzahl eingegeben und in der Variablen `g` gespeichert, dann der Teiler, der in der Variablen `t` gespeichert wird, anschließend wird das Ganzzahlergebnis mit dem Operator `//` berechnet und in `e` gespeichert, dann wird der Rest mit dem Operator `%` berechnet und in `r` gespeichert. Anschließend wird das Ergebnis (`e` und `r`) mit dem `print`-Befehl ausgegeben.

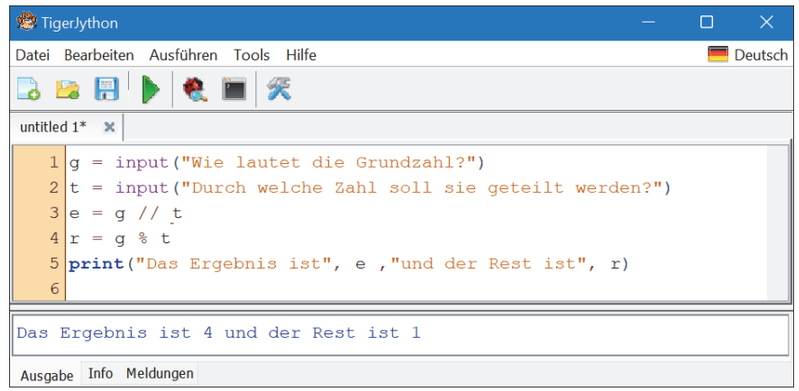


Abbildung 7.6 Das kommt heraus, wenn man 69 und 17 eingibt.

Alles, wofür es Formeln gibt, kann man auch direkt in ein hübsches Programm umwandeln. Dann übernimmt Python einfach die Berechnung, und selbst muss man nur noch die Ausgangswerte eingeben und erhält am Schluss sein fertiges Ergebnis. Anwendungsgebiete gibt es dafür unendlich viele, denn überall im Alltag müssen wir Dinge berechnen.

Die Zeit, die jemand bei bestimmter Geschwindigkeit für eine Strecke braucht, die Fläche einer Wand, von der man die Maße kennt, die Anzahl der Pixel auf einem Bildschirm mit bestimmter Auflösung, die Höhe der Mehrwertsteuer von einem bestimmten Betrag usw. Wenn dir etwas einfällt, das du immer wieder mühsam berechnen musst: Mach ein kleines Python-Programm daraus.

Das magische Quadrat

Hier kommt noch ein cooles Projekt zum Abschluss dieses Kapitels: das magische Quadrat.

Kennst du magische Quadrate? Das sind Quadrate, die zum Beispiel aus 16 Feldern (4 × 4) bestehen, und in jedem Feld steht eine Zahl. Die Summe der Zahlen jeder Reihe, waagerecht, senkrecht und diagonal, ist dabei genau gleich, ebenfalls die Summe der 2-×-2-Quadrate im Innern des magischen Quadrats sowie die Summe der vier Ecken.

Hier ist ein Beispiel:

2	1	12	7
11	8	1	2
5	10	3	4
4	3	6	9

Abbildung 7.7 Einfaches magisches Quadrat mit der Summe 22

Addierst du die Reihen, ist die Summe immer 22, ebenso bei allen Spalten, bei den beiden Diagonalen, bei den vier Viererquadraten, aus denen sich das Ganze zusammensetzt, und sogar beim Mittelquadrat. Auch wenn du die vier Ecken addierst, kommt 22 raus. Insgesamt erhältst du beim Zusammenzählen 18-mal auf verschiedene Weise die Summe 22. Magisch, oder?

Wenn du jetzt möchtest, dass Python dir so ein Quadrat erstellt – mit änderbaren anderen Zahlen als den abgebildeten –, was brauchst du dann dafür? Klar, eine Formel, mit der man die einzelnen Zahlen des magischen Quadrats errechnen kann. Formeln kann man nachschlagen – mit Google findet man fast alles. Du musst sie dir also nicht selbst ausdenken, du musst sie nur in deinem Programm richtig anwenden.

Das Erstellen eines magischen Quadrats funktioniert so: Du legst zwei ganze Zahlen fest, die du `a` und `b` nennst. Egal, welche, beide müssen mindestens 1 sein oder höher. Nun kannst du die Inhalte des Quadrats mit der folgenden Formel berechnen, und schon klappt es mit der Magie:

a+b	a	12*a	7*a
11*a	8*a	b	2*a
5*a	10*a	3*a	3*a+b
4*a	2*a+b	6*a	9*a

Abbildung 7.8 Das sind die Berechnungsformeln für jede Zelle des magischen Quadrats. a und b sind dabei beliebige ganze Zahlen über 0.

Und noch eine Formel gibt es dazu: Nachdem du die Zahlen `a` und `b` festgelegt hast, ergibt sich die magische Summe aller Reihen mit der Berechnung `21 * a + b`.

Mehr musst du gar nicht wissen. Du kannst jetzt ein Programm schreiben, das dir, nachdem du `a` und `b` festgelegt hast, alle 16 Zahlen des dazu berechneten magischen Quadrats ausgibt, in vier Reihen. Das Abtippen lohnt sich:

```
a = input("Gib einen Wert für a ein:")
b = input("Gib einen Wert für b ein:")
print("Die Summe aller Reihen, Spalten und Quadrate ist:",a*21+b)
# Magisches Quadrat ausgeben:
print("------------------------------------------------------------")
print(a+b,a,12*a,7*a)
print(11*a,8*a,b,2*a)
print(5*a,10*a,3*a,3*a+b)
print(4*a,2*a+b,6*a,9*a)
print("------------------------------------------------------------")
```

Erst wird also `a` eingegeben (eine beliebige Zahl), dann `b` (ebenfalls beliebig), dann errechnet Python erst einmal die magische Summe (mit der Formel `a * 21 + b`) und gibt sie aus, danach berechnet es nach unserer Formelvorgabe alle Zahlen für alle Kästchen und gibt sie in vier Reihen aus. Die ausgegebenen Striche dienen nur der Übersicht bei der Ausgabe und kennzeichnen den Anfang und das Ende der Zahlen.

Probiere es erst mal mit den Zahlen 1 und 1 für `a` und `b` aus – dann sollte genau das magische Quadrat, das weiter oben abgebildet ist, dabei herauskommen, mit der Summe 22. Wenn nicht, dann gibt es irgendwo einen Fehler in deinem Programm.

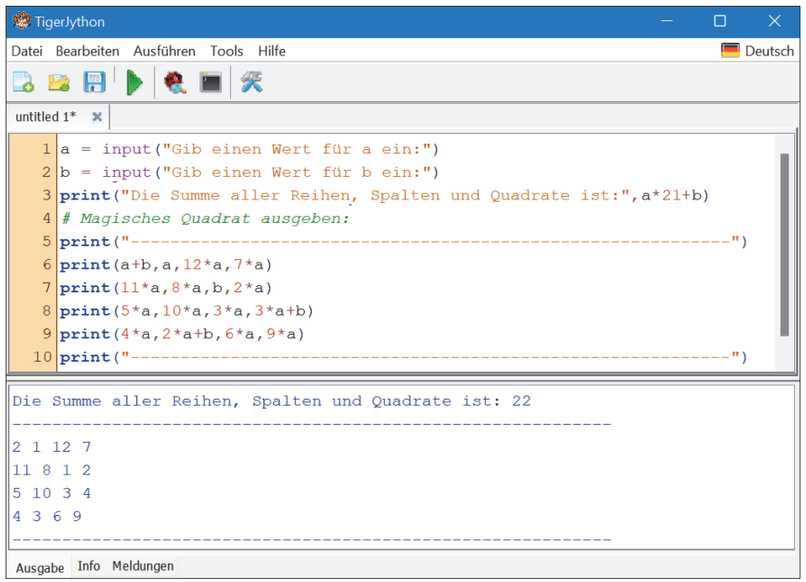


Abbildung 7.9 Das einfachste magische Quadrat (a = 1, b = 1) sollte so ausgegeben werden.

Jetzt kannst du beliebige Werte testen. Die Zahlen können hoch werden, und du kannst es im Kopf vielleicht gar nicht mehr nachrechnen – aber wenn das Programm korrekt eingegeben wurde, kannst du sicher sein, dass das magische Quadrat funktioniert.

Kommentare im Code mit »#«

Die vierte Zeile im Code hier ist übrigens eine Kommentarzeile: Du kannst in jedes Python-Programm jederzeit Kommentare einfügen, als eigene Zeile oder direkt nach einem Befehl. Sie beginnen immer mit einem #-Zeichen und enden am Ende der Zeile. Kommentare werden bei der Ausführung des Programms ignoriert. Sie dienen nur dazu, Erläuterungen und Anmerkungen ins Programm einzufügen.

Variation: Magisches Quadrat mit fester Summe

Nun wollen wir noch einen kleinen Schritt weiter gehen. Wie wäre es, ein magisches Quadrat zu erstellen, bei dem die Summe vorher feststeht? Du könntest dann zum Beispiel eine Person, die über 21 ist, nach ihrem Alter fragen und für sie ein ganz persönliches magisches Quadrat ihres Alters erstellen. Das macht Eindruck!

Wie müssten wir das Programm abändern?

Für diese Variante werden also nicht mehr die Werte `a` und `b` eingegeben, sondern es wird nur die gewünschte Summe eingegeben. Aus der Summe sollen jetzt `a` und `b` berechnet werden. Der Rest vom Programm bleibt gleich (die Ausgabe der Zahlen). Die Summe muss größer als 21 sein, denn 22 ist die Summe des kleinsten magischen Vierer-quadrats.

Wenn die Summe also `a*21` plus `b` ist, dann kann man umgekehrt auch `a` und `b` aus einer vorgegebenen Summe ermitteln. Wenn du gut in Mathe bist, kommst du vielleicht selbst drauf, ansonsten werde ich es dir sagen:

- `a` ist die Summe geteilt durch 21 (als ganze Zahl)
- `b` ist der Rest, der bleibt, wenn man die Summe durch 21 teilt.

So etwas haben wir doch schon mal gemacht – im Rechner mit Rest. Die Formel ist also genauso:

- `a = summe // 21` (Ganzzahldivision mit dem doppelten Schrägstrich)
- `b = summe % 21` (Rest mit Prozentzeichen ermitteln)

Achtung

Diese Methode funktioniert für alle Zahlen, die **nicht** Vielfache von 21 sind – also **nicht** für 21, 42, 63, 84 ..., weil dort die zweite Zahl 0 wäre. Man könnte diese Fälle im Programm gesondert behandeln – aber der Einfachheit halber belassen wir es jetzt einmal dabei.

Und so sieht dann unser geändertes Programm aus:

```
summe = input("Gib eine Summe (über 21) ein:")
# a und b berechnen:
a = summe // 21
b = summe % 21
# Magisches Quadrat ausgeben:
print("-------------------------------------------------------------")
print(a+b,a,12*a,7*a)
print(11*a,8*a,b,2*a)
print(5*a,10*a,3*a,3*a+b)
print(4*a,2*a+b,6*a,9*a)
print("-------------------------------------------------------------")
```

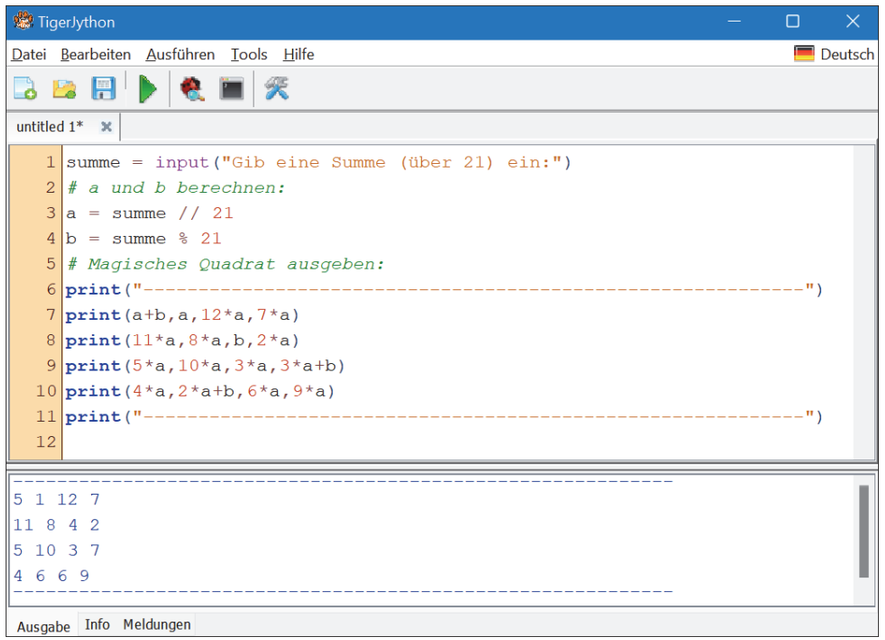


Abbildung 7.10 Magisches Quadrat für die Summe 25

Jetzt kannst du (fast) jede beliebige Summe von 22 bis unendlich eingeben, und Python wird dir daraus ein gültiges magisches Quadrat berechnen.

Zusammenfassung

- Programme sind eine Folge von Befehlen.
- Die meisten Programme bestehen aus den Elementen Dateneingabe, Datenverarbeitung (Berechnung), Datenausgabe.
- Beim Start eines Programms führt Python nacheinander jeden Befehl in der Liste aus, von oben nach unten. Wenn ein Befehl abgearbeitet worden ist, folgt der nächste.
- Zum Zwischenspeichern von Werten und Zeichenketten werden Variablen verwendet. Diese verwendet man genauso wie ihren Inhalt.
- Wenn du die Formel kennst, kannst du für jede Berechnung, egal, ob einfach oder kompliziert, ein Programm schreiben, das dir die Ermittlung des gewünschten Wertes vereinfacht.
- Du kannst in jedes Programm Kommentare einfügen, um sie besser verständlich zu machen. Kommentare beginnen mit einem #-Zeichen und enden am Ende der Zeile.

Kapitel 14

Sound programmieren

Bevor es an die Objektprogrammierung und dann an die ersten richtigen Spiele geht, wollen wir uns noch einmal mit Klang beschäftigen – denn durch Klänge werden Programme erst richtig lebendig!

Nicht nur Spiele, jedes Programm kann Klänge und Geräusche oder auch Musik abspielen, und manche können sogar sprechen. Ein Programm, das nicht nur fürs Auge ist, sondern das man auch hören kann, macht einfach mehr her.

Sound in Python abspielen

Alles, was du dazu brauchst, ist eine Klangdatei mit dem Sound, den du haben willst – selbst aufgenommen geht auch! Die kannst du dann in dein eigenes Programm einbinden. Und schon kann dein Programm mithilfe des in TigerJython enthaltenen Moduls `soundsystem` jedes beliebige Geräusch oder Musik und Sprache abspielen.

Dazu solltest du am besten etwas mehr darüber wissen, was eine Klangdatei auf dem Computer ist. Wenn du das alles schon weißt, kannst du den nächsten Abschnitt gern überspringen.

Was sind denn eigentlich Klangdateien?

Klangdateien sind *digitalisierte Geräusche, die zu einer Einheit aus Amplitudenwerten zusammengefasst wurden*. Um Klänge zu erzeugen, muss ein Lautsprecher in einer bestimmten Geschwindigkeit (das ist die Frequenz) hin- und herschwingen (die Stärke der Schwingung nennt man Amplitude). Wenn du vor einer empfindlichen Membran singst, dann wird diese je nachdem, welche Töne du singst, anfangen zu schwingen, je lauter, desto stärker, je höher, desto schneller. Wenn diese Membran ihre Schwingungen in elektrische Spannungen umsetzt, die man messen und aufzeichnen kann, hast du ein Mikrofon. Wenn ein Lautsprecher nun diese Schwingungen genauso, wie das

Mikrofon sie aufgezeichnet hat, wieder in seinen Membranen erzeugen kann, dann wird eine Aufnahme abgespielt. Das war früher analog: Auf Schallplatten hat eine Nadel während der Schwingungen mehr oder weniger tief in das Vinyl geritzt – auf Tonbändern werden die Schwingungen, während das Band vorbeiläuft, durch unterschiedlich starke Magnetisierung festgehalten.

Digital läuft das alles prinzipiell genauso – aber die Schwingungen des Mikrofons oder Aufnahmegeräts werden bei der Aufnahme in Zahlen umgesetzt, die in extrem schneller Folge, zum Beispiel 44.100-mal pro Sekunde, gemessen und gespeichert werden (das entspricht dem Standard einer Musik-CD – 44.1 kHz). Das heißt: 1 Sekunde Klang besteht aus 44.100 Zahlen, die nacheinander genau beschreiben, wie weit die Membran in welche Richtung ausschlägt. Wenn der Computer diese 44.100 Zahlen nun in der gleichen Geschwindigkeit wieder an den Lautsprecher sendet und dieser die Zahlen wieder in Membranausschläge umsetzt, ertönt 1 Sekunde lang wieder exakt der aufgenommene Klang.

Das Dateiformat, bei dem jede Zahl genau einem Membranzustand entspricht, nennt man auch WAV-Format – das steht für *wave* (Welle). Klangdateien im WAV-Format erkennst du in Windows an der Endung *.wav* – sie waren früher die Standarddateien, um Klänge zu speichern oder abzuspielen.

Da WAV-Dateien aber bei langen Musikstücken sehr viel Speicherplatz einnehmen (wie gesagt: für jede Sekunde 44.100 Zahlen und für Stereo genau doppelt so viel, denn da spielen ja zwei Lautsprecher unterschiedliche Versionen des Klanges ab), hat man zunächst vor allem für die Übertragung über das Internet irgendwann ein Format erfunden, das den (nahezu) gleichen Klang mit viel weniger Daten abspielen kann. Die Klangdaten werden komprimiert – wiederkehrende Muster in den Daten werden zusammengefasst gespeichert, und mit einem komplizierten Verfahren werden die Klangdaten intern genauso weit vereinfacht, dass das menschliche Ohr den Unterschied praktisch nicht wahrnimmt. Dieses Format wurde unter dem Namen *MP3* bekannt. Heute ist das längst der Standard für Audiodateien. Der Computer oder das Gerät, das diese Dateien abspielt, muss sie intern erst einmal wieder dekomprimieren, das heißt, aus einer MP3-Datei wird intern wieder eine WAV-Datei gemacht, die dann (über die Soundkarte) an den Lautsprecher gesendet wird.

Egal, ob du WAV-Dateien oder MP3-Dateien hast: Beides kannst du in TigerJython sehr leicht in deine Programme einbinden und abspielen.

WAV-Dateien abspielen

Um überhaupt Klangdateien in einem Programm verwenden zu können, musst du immer als Erstes das Modul `soundsystem` einbinden. Das geschieht am einfachsten über den folgenden Befehl:

```
from soundsystem import *
```

Das Abspielen ist nun ganz einfach. Als Erstes wird der Soundplayer aktiviert, und es wird ihm eine Datei gegeben, die er zum Abspielen bereit macht. Danach wird der Player gestartet.

```
openSoundPlayer("klangdatei.wav")
play()
```

Das war's – damit kannst du die Datei namens *klangdatei.wav* abspielen.

Du hast gerade keine WAV-Datei zur Hand? Zum Glück hat TigerJython schon ein paar kurze Dateien für Geräusche eingebaut. Mit denen kannst du es ganz einfach testen:

```
from soundsystem import *
openSoundPlayer("wav/bird.wav")
play()
```

Starte das Programm und hör hin: Ein kurzes Vogelgezwitscher ist (nach einer kurzen Initialisierungszeit) zu hören. Es hat geklappt.

Probiere noch ein paar andere eingebaute Klänge aus, die in TigerJython enthalten sind. Du musst nur den Namen der Klangdatei ändern:

```
wav/boing.wav wav/cat.wav wav/click.wav wav/explode.wav wav/frog.wav wav/
mmm.wav wav/notify.wav wav/ping.wav
```

Davon kannst du vielleicht schon etwas für eigene Programme gebrauchen.

Wenn du deine eigenen WAV-Klangdateien verwenden willst, dann erstelle am besten einen Ordner namens *wav* dort, wo auch deine Programme gespeichert sind, und kopiere deine Klangdateien dort hinein. Nun kannst du sie auch, wie oben gezeigt, per `soundsystem` abspielen, indem du `"wav/DeinDateiname.wav"` einsetzt.

MP3-Dateien abspielen

Ich nehme an, du hast Musikdateien (wenn du welche hast) vielleicht eher im Format MP3 bei dir vorliegen? Das Abspielen von MP3-Dateien funktioniert prinzipiell genauso wie das Abspielen von WAV-Dateien. Nur muss das Programm natürlich wissen, dass es eine Datei im MP3-Format erhält und diese erst einmal decodieren muss. Deshalb werden MP3-Dateien mit einer anderen Funktion geöffnet:

```
from soundsystem import *

openSoundPlayerMP3("mp3/meinSong.mp3")
play()
```

Du brauchst dafür nur eine MP3-Datei, zum Beispiel deinen Lieblingssong, oder eine beliebige Datei, wie du sie massenweise legal aus dem Internet herunterladen kannst. Leg dir in deinem Python-Ordner ein Verzeichnis namens *mp3* an und kopiere deine Datei dorthin. Nun kannst du sie mit Python abspielen. Allerdings musst du das Programm zuvor erst einmal im Python-Ordner speichern, damit Python den MP3-Ordner findet.

Weitere Befehle für den Sound-Player

Neben der Funktion `openSoundPlayer()` und `openSoundPlayerMP3()` zum Laden der Datei und dem Befehl `play()` zum Abspielen der geladenen Datei gibt es noch zahlreiche weitere Befehle im `soundsystem`, die du später in deinen Programmen verwenden kannst. Hier nur als Einblick die drei wichtigsten:

- `pause()` setzt das Abspielen auf Pause, ein folgendes `play()` macht dort weiter, wo pausiert wurde.
- `stop()` beendet das Abspielen und setzt die Datei wieder auf Anfang.
- `setVolume(v)` – damit wird die Lautstärke des Players eingestellt: 0 ist stumm, 1.000 ist das Maximum.

Eigene Musik machen

Du kannst in TigerJython sogar ganz eigene Musik machen. Dafür brauchst du nicht einmal das Modul `soundsystem`, denn es gibt schon den eingebauten Befehl `playTone()`, der Noten spielen kann. Kannst du Noten?

Probiere mal das hier aus:

```
playTone("cdefgfedc",200, instrument="harp")
```

Wenn du das Programm startest, hörst du eine Tonleiter, die von C bis G hochgeht und dann wieder herunter.

Der erste Wert gibt die zu spielenden Noten an (Kleinbuchstaben: mittlere Oktave, Großbuchstaben: tiefere Oktave, mit Apostroph: höhere Oktave), der zweite Wert (200) die Dauer jedes Tons in Millisekunden, mit dem dritten Wert (den man auch weglassen kann) wird das Instrument gewählt. `"harp"` klingt auf den meisten Systemen ganz angenehm.

Die Klänge werden über den MIDI-Standard abgespielt, der im Betriebssystem des Computers enthalten ist. Dementsprechend ist die Qualität der Instrumente meist niedrig, aber trotzdem einigermaßen erkennbar. Als Namen der Instrumente kannst du die Standard-MIDI-Namen wie *piano, guitar, harp, trumpet, xylophone, organ, violin, panflute, bird* usw. einsetzen.

Du kannst sogar Töne in unterschiedlichen Längen in einer Liste zusammenfassen, die dann insgesamt abgespielt wird:

```
playTone([("cdeccdecef",300),("g",600),("ef",300),("g",600)], instrument =
 "harp")
```

Damit wird der Anfang des Kanons »Bruder Jakob« gespielt. Die Liste, die `playTone()` braucht, ist eine Tupel-Liste (wir haben dieses Format bisher nicht verwendet) – aber es geht ganz einfach:

```
playTone ( [("töne",dauer), ("töne",dauer) , (… usw. … , usw. …) ] , instrument=
"harp" )
```

Innerhalb der Funktionsklammern gibt es also zwei eckige Klammern, die die Ton-Dauer-Paare einschließen. Jede Tonfolge wird dann mit einer runden Klammer begonnen, danach folgt ein Komma, danach die Länge der Töne als Zahl, bevor die runde Klammer wieder geschlossen wird. Nun folgen wieder ein Komma und die nächste Tonfolge samt Dauer in runden Klammern. Am Schluss wird die eckige Klammer geschlossen, gefolgt von einem Komma, `instrument = "instrument"`, bevor die Funktionsklammer geschlossen wird. Das Instrument muss hier also nur einmal ganz am Schluss deklariert werden und gilt für alle Töne in den eckigen Klammern zuvor.

So sieht das ganze Lied dann aus:

```
playTone([("cdeccdecef",300),("g",600),("ef",300),("g",600),("gagf",150),
("ec",300),("gagf",150),("ec",300),("cG",300),("c",600),("cG",300),("c",600)],
instrument = "harp")
```

Klar, anspruchsvolle Musiker kommen damit nicht weit. Dafür ist es zu unpräzise, und die Klänge sind qualitativ zu niedrig. Aber du kannst einfache Tonfolgen durchaus mal in einem Spiel oder Quiz oder einem anderen Programm einsetzen.

Zum Beispiel so:

```
print("Du hast gewonnen!")
playTone ("cegc'",100,instrument="harp")
```

Aufgabe

Baue kleine Tonfolgen in Programme ein, die du schon geschrieben hast!

Sprachsynthese – lass den Computer sprechen!

Jetzt wird es richtig cool! Mit der Bibliothek `soundsystem` und weiteren in TigerJython bereits eingebauten Komponenten kannst du den Computer auch richtig sprechen lassen, und zwar jeden Text, den du möchtest!

Probiere mal das hier aus:

```
from soundsystem import *
initTTS() # Text To Speech starten
selectVoice("german-man") # Stimme wählen
voice = generateVoice("Hallo, ich kann sprechen!")
openSoundPlayer(voice)
play()
```

Nach kurzer Initialisierungszeit sagt der Computer deutlich: »Hallo, ich kann sprechen!«

Du brauchst dazu das Soundsystem, das ganz normal importiert wird, dann musst du das TTS -System (»Text To Speech«) starten, das geht mit dem Befehl `initTTS()`. Anschließend muss eine Stimme gewählt werden, das geht mit `selectVoice("stimme")` – statt »german-man« kannst du gerne auch »german-woman« wählen. Jetzt musst du

den Klang deines Textes erzeugen. Das machst du mit der Funktion `generateVoice()` – das Ergebnis davon öffnest du im Soundplayer und startest ihn, wie die WAV-Dateien auch, nur dass statt einer Datei eine selbst erzeugte Stimmausgabe gespielt wird.

Die ersten drei Befehle brauchen nur einmal am Anfang des Programms ausgeführt zu werden. Danach kannst du die Stimme mit beliebigen Inhalten immer wieder mit den drei Befehlen

```
voice = generateVoice("Text oder Variable")
openSoundPlayer (voice)
play()
```

abspielen.

Wie wäre es mit einem sprechenden Additionsrechner?

```
from soundsystem import *
initTTS()
selectVoice("german-man")

x = input("Gib eine Zahl ein:")
y = input("Jetzt die zweite:")
rechnung = str(x)+" plus "+str(y)+" = "+str(x+y)

voice = generateVoice(rechnung)
openSoundPlayer(voice)
play()
```

Du gibst nacheinander zwei Zahlen ein, und der Computer spricht die Additionsaufgabe samt Ergebnis!

Auch in anderen Programmen kann Sprache den Reiz deutlich erhöhen. Zum Beispiel das Zahlenratespiel: Wie wäre es, wenn der Computer dir nur sagt, ob die Zahl zu klein oder zu groß ist oder stimmt? Viel lustiger! So würde das Zahlenratespiel mit Stimmausgabe aussehen:

```
import random
from soundsystem import *
initTTS()
selectVoice("german-man")
```

```
zufallszahl = random.randint(1,100)
eingabe = 0
while eingabe != zufallszahl:
    eingabe = input("Rate die Zahl:")
    if eingabe > zufallszahl:
        voice = generateVoice(str(eingabe)+" ist zu groß.")
    if eingabe < zufallszahl:
        voice = generateVoice(str(eingabe)+" ist zu klein.")
    openSoundPlayer(voice)
    play()

voice = generateVoice("Glückwunsch, "+str(eingabe)+" ist die richtige Zahl!")
openSoundPlayer(voice)
play()
```

Auf einen Blick

Inhalt

22 Tic Tac Toe – Brettspiele mit gamegrid 325

23 Wie geht es weiter? 351

Materialien zum Buch

Auf der Website zu diesem Buch stehen folgende Materialien bereit:

- **Die Entwicklungsumgebung TigerJython**
- **Alle Codebeispiele aus dem Buch**

Gehe dazu auf *www.rheinwerk-verlag.de/5399*.

Klicke auf den Reiter MATERIALIEN. Du siehst die Dateien zum Herunterladen mit einer kurzen Beschreibung. Klicke auf den Button HERUNTERLADEN, um den Download zu starten. Es kann einige Zeit dauern, bis der Download abgeschlossen ist.