dpunkt.verlag

Al Sweigart

Eigene Spiele programmieren: Python lernen

Der spielerische Weg zur Programmiersprache

Was sind E-Books von dpunkt?

Unsere E-Books sind Publikationen im PDF- oder ePub-Format, die es Ihnen erlauben, Inhalte am Bildschirm zu lesen, gezielt nach Informationen darin zu suchen und Seiten daraus auszudrucken.

Sie benötigen zum Ansehen den Acrobat Reader oder ein anderes adäquates Programm bzw. einen E-Book-Reader.

E-Books können Bücher (oder Teile daraus) sein, die es auch in gedruckter Form gibt (bzw. gab und die inzwischen vergriffen sind). (Einen entsprechenden Hinweis auf eine gedruckte Ausgabe finden Sie auf der entsprechenden E-Book-Seite.)

Es können aber auch Originalpublikationen sein, die es ausschließlich in E-Book-Form gibt. Diese werden mit der gleichen Sorgfalt und in der gleichen Qualität veröffentlicht, die Sie bereits von gedruckten dpunkt.büchern her kennen.

Was darf ich mit dem E-Book tun?

Wie erhalte ich das E-Book von dpunkt?

Sobald der Bestell- und Bezahlvorgang abgeschlossen ist, erhalten Sie an die von Ihnen angegebene Adresse eine Bestätigung. Außerdem erhalten Sie von dpunkt eine E-Mail mit den Downloadlinks für die gekauften Dokumente sowie einem Link zu einer PDF-Rechnung für die Bestellung.

Die Links sind zwei Wochen lang gültig. Die Dokumente selbst sind mit Ihrer E-Mail-Adresse und Ihrer Transaktionsnummer als Wasserzeichen versehen.

Wenn es Probleme gibt?

Bitte wenden Sie sich bei Problemen an den dpunkt.verlag
e-mail: ebooks@dpunkt.de
fon: 06221/1483-0.

Al Sweigart

Eigene Spiele programmieren – Python lernen

Der spielerische Weg zur Programmiersprache

dpunkt.verlag

Al Sweigart

Lektorat: Gabriel Neumann
Fachgutachterin (amerikanische Ausgabe): Ari Lacenski
Copy-Editing: Claudia Lötschert
Übersetzung & Satz: G&U Language & Publishing Services GmbH, www.gundu.com
Herstellung: Nadine Thiele
Umschlaggestaltung: Helmut Kraus, www.exclam.de
nach der Originalvorlage von No Starch Press
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Print 978-3-86490-492-9
PDF 978-3-96088-322-7
ePub 978-3-96088-323-4
mobi 978-3-96088-324-1

1. Auflage 2017

5 4 3 2 1 0

Inhalt

Für Caro

Der Autor

Al Sweigart ist Softwareentwickler, Fachbuchautor und ein echter Frood, der weiß, wo sein Handtuch ist. Er hat mehrere Programmierbücher für Einsteiger geschrieben, darunter *Routineaufgaben mit Python automatisieren* und *Coole Spiele mit Scratch*, die ebenfalls beim dpunkt.verlag erschienen sind. Auf seiner Website *https://inventwithpython.com/* stehen seine Bücher unter einer Creative-Commons-Lizenz auch kostenlos zur Verfügung.

Die Fachgutachterin der amerikanischen Ausgabe

Ari Lacenski ist Entwicklerin für Android-Anwendungen und Python-Software. Sie lebt in Bay Area, wo sie auf *http://gradlewhy.ghost.io/* über Android-Programmierung schreibt und als Mentorin für Women Who Code dient.

Danksagung

Ohne die herausragende Arbeit des Teams von No Starch Press wäre dieses Buch nicht möglich gewesen. Ich danke meinem Herausgeber Bill Pollock und meinen Lektoren Laurel Chun, Jan Cash und Tyler Ortman für ihre unglaubliche Hilfe während des gesamten Vorgangs, meiner Fachgutachterin Ari Lacenski für ihre gründliche Durchsicht und Josh Ellingson für ein weiteres hervorragendes Titelbild.

Einleitung

Die ersten Videospiele, die ich im Kindesalter spielte, machten mich süchtig. Ich wollte sie nicht nur spielen, ich wollte selbst so etwas gestalten. Damals fand ich ein Buch, das mir zeigte, wie ich eigene erste Programme und Spiele schreiben konnte. Das machte Spaß und war ganz einfach. Die ersten Spiele, die ich entwickelte, waren ähnlich wie diejenigen in diesem Buch. Sie waren nicht so ausgefeilt wie die Nintendo-Spiele, die meine Eltern für mich kauften, aber sie waren mein eigenes Werk.

Jetzt, als Erwachsener, habe ich immer noch Spaß am Programmieren und werde sogar dafür bezahlt. Doch auch wenn Sie nicht vorhaben, Programmierer zu werden, ist Programmierung eine nützliche und unterhaltsame Fähigkeit. Sie schulen sich dabei, logisch zu denken, planvoll vorzugehen und ihre Ideen kritisch zu betrachten, wenn Sie Fehler in Ihrem Code gefunden haben.

Viele Programmierbücher für Einsteiger fallen in eine von zwei verschiedenen Kategorien. Die erste Art von Büchern konzentriert sich so sehr auf »Spielentwicklungssoftware« oder stark vereinfachende Sprachen, dass man das, was da gelehrt

wird, gar nicht mehr als »Programmieren« bezeichnen kann. Die zweite Kategorie umfasst Bücher, die Programmierung in der Art eines Mathebuchs abhandeln: lauter Lehrsätze und Begriffe, aber nur wenige praktische Anwendungen für die Leser. Das vorliegende Buch verfolgt einen anderen Ansatz und bringt Ihnen das Programmieren am Beispiel von Videospielen bei. Ich stelle Ihnen den Quellcode der Spiele jeweils zu Anfang vor und erkläre dann die Programmierprinzipien anhand dieser Beispiele. Diese Vorgehensweise war für mich selbst von entscheidender Bedeutung, als ich Programmieren lernte. Je mehr ich darüber erfuhr, wie die Programme anderer Leute funktionierten, umso mehr Ideen bekam ich für meine eigenen.

Sie brauchen nicht mehr als einen Computer, den Python-Interpreter (eine kostenlose Software) und dieses Buch. Wenn Sie gelernt haben, wie Sie die Spiele in diesem programmieren, können Sie auch Ihre eigenen entwickeln.

Computer sind unglaubliche Maschinen. Zu lernen, wie man sie programmiert, ist gar nicht so schwer, wie die meisten Leute glauben. Ein Computer*programm* ist eine Folge von Anweisungen, die ein Computer verstehen kann, ähnlich wie ein Buch eine Folge von Sätzen ist, die der Leser verstehen kann. Um einen Computer anzuweisen, etwas zu tun, müssen Sie ein Programm in einer Sprache schreiben, die er versteht. In diesem Buch lernen Sie die Programmiersprache Python kennen. Es gibt noch viele andere Programmiersprachen, die Sie lernen können, z. B. BASIC, Java, JavaScript, PHP oder C++.

Als Kind habe ich BASIC gelernt, aber neuere Programmiersprachen wie Python lassen sich noch einfacher lernen. Python wird auch von professionellen Programmierern sowohl bei der Arbeit als auch für Hobbyprojekte verwendet. Installation und Verwendung sind außerdem völlig kostenlos. Sie brauchen lediglich eine Internetverbindung, um die Sprache herunterzuladen.

Da Videospiele nichts anderes sind als Computerprogramme, bestehen sie ebenfalls aus Anweisungen. Im Vergleich zu den Spielen für die Xbox, die PlayStation oder den Nintendo wirken diejenigen, die Sie in diesem Buch erstellen werden, ziemlich simpel. Sie weisen keine anspruchsvolle Grafik auf, sondern sind absichtlich einfach gehalten, sodass Sie sich drauf konzentrieren können, Programmieren zu lernen. Außerdem müssen Spiele nicht kompliziert sein, um Spaß zu machen.

Zielgruppe

Programmieren an sich ist nicht schwer. Dagegen kann es durchaus schwer sein, Anleitungen zu finden, die einem sagen, wie man interessante Dinge programmiert. Andere Computerbücher behandeln viele Themen, die Neulinge gar nicht brau-

chen. Dieses Buch zeigt Ihnen, wie Sie Ihre eigenen Spiele programmieren. Damit erwerben Sie eine nützliche Fähigkeit und haben außerdem unterhaltsame Spiele, die Sie stolz vorstellen können. Dieses Buch richtet sich an folgende Leser:

- Anfänger ohne jegliche Vorkenntnisse, die sich selbst das Programmieren beibringen möchten
- Kinder und Jugendliche, die durch das Schreiben von Spielen programmieren lernen möchten
- Erwachsene und Lehrer, die anderen Programmieren beibringen möchten
- Jegliche Personen, ob jung oder alt, die anhand einer professionellen Programmiersprache programmieren lernen wollen

Der Aufbau dieses Buches

In den meisten Kapiteln dieses Buches wird jeweils ein neues Spielprojekt vorgestellt und erläutert. Einige wenige Kapitel sind zusätzlichen interessanten Themen gewidmet, z. B. dem Debugging. Neue Programmierprinzipien werden jeweils dann vorgestellt, wenn sie in den Spielen vorkommen. Die Kapitel sollten in der vorliegenden Reihenfolge gelesen werden. Die folgende kurze Aufstellung zeigt, was Sie in den einzelnen Kapiteln erwartet:

Kapitel 1: Die interaktive Shell erklärt, wie Sei die interaktive Shell von Python verwenden können, um Code zeilenweise auszuprobieren.

Kapitel 2: Programme schreiben erklärt, wie Sie im Dateieditor von Python vollständige Programme schreiben.

In **Kapitel 3: Zahlen raten** programmieren Sie das erste Spiel in diesem Buch. Dabei muss der Spieler eine Geheimzahl raten und erhält jeweils den Hinweis, ob seine Vermutung zu hoch oder zu niedrig war.

In **Kapitel 4: Ein Kalauerprogramm** schreiben Sie ein einfaches Programm, das dem Benutzer mehrere Flachwitze erzählt.

In **Kapitel 5: Im Reich der Drachen** programmieren Sie ein Ratespiel, in dem der Spieler zwischen zwei Höhlen wählen muss: In einer der beiden wohnt ein freundlicher Drache, in der anderen ein hungriger.

Kapitel 6: Der Debugger erklärt, wie Sie Probleme in Ihrem Code mithilfe des Debuggers lösen.

Kapitel 7: Galgenmännchen: Entwurf mit einem Flussdiagramm beschreibt, wie Sie Flussdiagramme zur Planung anspruchsvollerer Programme einsetzen, hier für das in den folgenden Kapiteln beschriebene Galgenmännchen-Spiel.

In **Kapitel 8: Galgenmännchen: Der Code** schreiben Sie das Galgenmännchen-Spiel anhand des Flussdiagramms aus Kapitel 7.

In **Kapitel 9: Galgenmännchen: Erweiterungen** erweitern Sie das Galgenmännchen-Spiel, indem Sie den Python-Datentyp der Dictionarys verwenden.

In **Kapitel 10: Tic-Tac-Toe** lernen Sie, wie Sie ein Tic-Tac-Toe-Spiel schreiben, bei dem eine künstliche Intelligenz gegen den menschlichen Benutzer antritt.

In **Kapitel 11: Bagels** schreiben Sie das Denksportspiel Bagels, in dem der Spieler anhand von Hinweisen eine Geheimzahl erraten muss.

Kapitel 12: Kartesische Koordinaten erklärt das kartesische Koordinatensystem, das Sie in nachfolgenden Spielen verwenden werden.

In **Kapitel 13: Sonar-Schatzsuche** schreiben Sie ein Spiel, in dem Sie das Meer nach verlorenen Schatztruhen absuchen.

In **Kapitel 14: Die Caesar-Chiffre** schreiben Sie ein einfaches Verschlüsselungsprogramm, mit dem Sie Geheimnachrichten verschlüsseln und entschlüsseln können.

In **Kapitel 15: Reversegam** programmieren Sie ein anspruchsvolles Reversi-ähnliches Spiel mit einer fast unschlagbaren künstlichen Intelligenz als Gegner.

Kapitel 16: Reversegam: AI-Simulation erweitert das Reversegam-Speil aus Kapitel 15, sodass nun verschiedene KIs gegeneinander antreten.

Kapitel 17: Grafik einsetzen führt das Python-Modul `pygame` ein und zeigt, wie Sie damit 2-D-Grafiken zeichnen können.

Kapitel 18: Grafiken animieren zeigt, wie Sie mit `pygame` Grafiken animieren.

In **Kapitel 19: Kollisionserkennung** lernen Sie, wie Sie in 2-D-Spielen Zusammenstöße zwischen Objekten erkennen können.

In **Kapitel 20: Sounds und Bilder verwenden** erweitern Sie Ihre einfachen `pygame`-Spiele um Klänge und Bilder.

Kapitel 21: Dodger mit Ton und Grafik wendet den Stoff aus den Kapiteln 17 bis 20 für ein animiertes Spiel namens Dodger an.

Die Codebeispiele

Die meisten Kapitel dieses Buches beginnen mit einem Beispieldurchlauf des Programms, das in dem Kapitel vorgestellt wird. Dieser Durchlauf zeigt, wie das Programm aussieht, wenn Sie es ausführen. Die Eingaben des Benutzers sind durch Fettdruck gekennzeichnet.

Ich rate Ihnen, den Code der einzelnen Programme jeweils selbst in den IDLE-Dateieditor einzugeben, anstatt ihn einfach nur herunterzuladen und zu kopieren. Wenn Sie sich die Zeit nehmen, den Code abzutippen, werden Sie mehr davon im Kopf behalten.

Zeilennummern und Einrückungen

Beim Abtippen des Quellcodes aus dem Buch dürfen Sie die Nummern zu Beginn der einzelnen Zeilen *nicht* mit eingeben! Nehmen wir an, Sie sehen die folgende Codezeile:

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

Hier dürfen Sie weder `9.` noch das darauffolgende Leerzeichen eingeben, sondern nur Folgendes:

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

Die Zeilennummern stehen hier nur, um in den Erklärungen auf die einzelnen Zeilen verweisen zu können. Sie gehören nicht zum Quellcode des Programms.

Abgesehen von diesen Zeilennummern jedoch müssen Sie den Code genau so eingeben, wie er in dem Buch erscheint. Einige Zeilen sind mit vier, acht oder mehr Leerzeichen eingerückt. Diese Leerzeichen zu Anfang einer Zeile wirken sich darauf aus, wie Python die Anweisungen deutet, weshalb es sehr wichtig ist, sie einzuschließen.

Betrachten Sie dazu das folgende Beispiel. Die Leerzeichen sind durch dicke Punkte (•) dargestellt, sodass Sie sie sehen können:

```
while guesses < 10:
····if number == 42:
········print('Hello')
```

Die erste Zeile ist nicht eingerückt, die zweite um vier und die dritte um acht Leerzeichen. In den tatsächlichen Beispielen in dem Buch sind die Leerzeichen zwar nicht durch solche Punkte gekennzeichnet, aber da alle Zeichen in IDLE die gleiche Breite aufweisen, können Sie die Anzahl der Leerzeichen einfach dadurch

bestimmen, dass Sie die Anzahl der Zeichen in der darüber- oder darunterliegenden Zeile zählen.

Lange Codezeilen

Manche Codeanweisungen sind so lang, dass sie nicht mehr in eine Buchzeile passen und daher in die nächste Codezeile umbrechen. Allerdings passt die Zeile auf den Computerbildschirm. Geben Sie sie dort in einer einzigen Zeile ein, ohne zwischendurch die Eingabetaste zu drücken. Ob eine Anweisung einfach nur fortgesetzt wird oder ob tatsächlich eine neue Anweisung beginnt, können Sie anhand der Zeilennummern auf der linken Seite erkennen. Das folgende Beispiel zeigt nur zwei Anweisungen:

```
1. print('This is the first instruction!xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
     xxxxxxxxxxxx')
2. print('This is the second instruction, not the third instruction.')
```

Die erste Anweisung bricht in die zweite Buchzeile um. Da die zweite Buchzeile keine Zeilennummer trägt, können Sie erkennen, dass es sich immer noch um die erste Codezeile handelt.

Python herunterladen und installieren

Um die Programme eingeben und ausprobieren zu können, Sie müssen eine Software installieren, die als Python-*Interpreter* bezeichnet wird und die Anweisungen versteht, die Sie in der Sprache Python geschrieben habe. Den Python-Interpreter werde ich im Folgenden der Einfachheit halber auch einfach als Python bezeichnen.

In diesem Abschnitt zeige ich Ihnen, wie Sie Python 3 – genauer gesagt, Python 3.4 – für Windows, OS X und Ubuntu herunterladen. Es gibt schon neuere Versionen als 3.4, aber das Modul `pygame`, das wir in den Kapiteln 17 bis 21 verwenden, unterstützt zurzeit nur die Versionen bis 3.4.

Beachten Sie auch, dass es erhebliche Unterschiede zwischen Python 2 und Python 3 gibt. Die Programme in diesem Buch verwenden Python 3. Wenn Sie versuchen, Sie mit Python 2 auszuführen, erhalten Sie Fehlermeldungen. Das ist so wichtig, dass ich bei jedem Programm den kleinen Pinguin hinzugefügt habe, um Sie darauf aufmerksam zu machen.

Wenn Sie mit Windows arbeiten, laden Sie den MSI-Installer für Windows x86-64 von *https://www.python.org/downloads/release/python-344/* herunter und doppelklicken darauf. Möglicherweise müssen Sie das Administratorkennwort für Ihren Computer eingeben.

Folgen Sie den Anweisungen, die der Installer anzeigt, um Python zu installieren. Die Vorgehensweise ist wie folgt:

1. Wählen Sie *Install for All Users* und klicken Sie auf *Weiter*.
2. Bestätigen Sie die Installation im Ordner *C:\Python34*, indem Sie auf *Weiter* klicken.
3. Klicken Sie auf *Weiter*, um den Abschnitt *Customize Python* zu überspringen.

Wenn Sie mit OS X arbeiten, laden Sie den MSI-Installer für Mac OS X 64 Bit/32 Bit von *https://www.python.org/downloads/release/python-344/* herunter und doppelklicken darauf. Möglicherweise müssen Sie das Administratorkennwort für Ihren Computer eingeben.

Folgen Sie den Anweisungen, die der Installer anzeigt, um Python zu installieren. Die Vorgehensweise ist wie folgt:

1. Wenn Sie eine Warnung der Art »Python.mpkg kann nicht geöffnet werden, da es von einem nicht verifizierten Entwickler stammt« erhalten, klicken Sie bei gedrückter ctrl-Taste auf die Datei *Python.mpkg* und wählen *Öffnen* aus dem dann angezeigten Menü. Möglicherweise müssen Sie das Administratorpasswort für Ihren Computer eingeben.
2. Klicken Sie im Willkommensabschnitt auf *Continue* und dann auf *Agree*, um die Lizenzverarbeitung zu akzeptieren.
3. Wählen Sie *Macintosh HD* (bzw. den Namen Ihrer Festplatte) aus und klicken Sie auf *Install*.

Wenn Sie Ubuntu verwenden, könne Sie Python wie folgt aus dem Ubuntu Software Center heraus installieren:

1. Öffnen Sie das Ubuntu Software Center.
2. Geben Sie in das Suchfeld oben rechts in dem Fenster **Python** ein.
3. Wählen Sie *IDLE (Python 3.4 GUI 64 bit)* aus.
4. Klicken Sie auf *Install*. Möglicherweise müssen Sie das Administratorpasswort Computer eingeben, um die Installation abzuschließen.

Sollten die angegebenen Vorgehensweisen nicht funktionieren, finden Sie alternative Installationsanweisungen für Python 3.4 auf *https://www.nostarch.com/inventwithpython/*.

IDLE starten

IDLE steht für Interactive Development Enrivonment, also »interaktive Entwicklungsumgebung«. Es handelt sich dabei um eine Art Textverarbeitungssystem zum Schreiben von Python-Programmen. Je nach Betriebssystem müssen Sie zum Starten dieser Software jeweils unterschiedlich vorgehen:

- Unter Windows klicken Sie auf das Startmenü unten links, geben **IDLE** ein und wählen *IDLE (Python GUI)*.
- Unter OS X öffnen Sie den Finder und klicken auf *Programme*. Doppelklicken Sie anschließend auf *Python 3.x* und dann auf das IDLE-Symbol.
- Unter Ubuntu und anderen Linux-Distributionen öffnen Sie ein Terminal-Fenster und geben **idle3** ein. Es kann auch sein, dass Sie auf *Applications* am oberen Bildschirmrand und dann auf *Programming* und *IDLE3* klicken können.

Das Fenster, das Sie beim Start von IDLE sehen, ist die *interaktive Shell* (siehe Abbildung). Wenn Sie hinter der Eingabeaufforderung >>> Python-Anweisungen in die Shell eingeben, führt Python sie aus. Anschließend erscheint wieder die Eingabeaufforderung >>>, um auf Ihre nächste Anweisung zu warten.

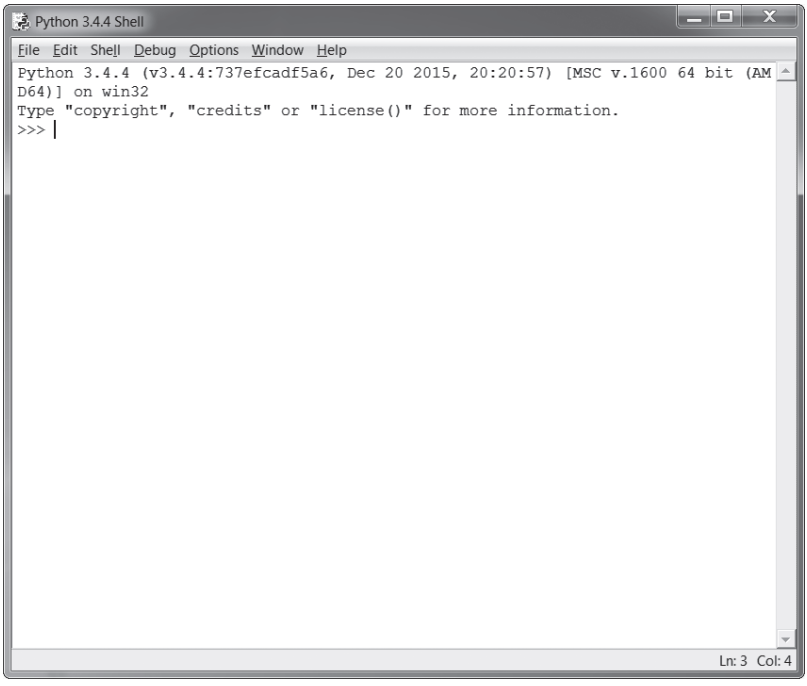


Abb. E–1 *Die interaktive Shell von IDLE*

Online Hilfe finden

Die Quellcodedateien und andere Materialien zu diesem Buch finden Sie auf *https://www.nostarch.com/inventwithpython/*. Wenn Sie Fragen zur Programmierung im Zusammenhang mit diesem Buch haben, können Sie auf *https://reddit.com/r/inventwithpython/* nachschauen oder sich über *al@inventwithpython.com* an mich wenden (in englischer Sprache).

Bevor Sie Ihre Fragen stellen, sollten Sie aber Folgendes unternehmen:

- Wenn Sie bei einem Programm aus diesem Buch, das Sie abgetippt haben, Fehlermeldungen erhalten, prüfen Sie, ob sich Tippfehler eingeschlichen haben. Das können Sie mit dem Vergleichswerkzeug (Diff Tool) machen, das auf *https://www.nostarch.com/inventwithpython#diff* zur Verfügung steht. Kopieren Sie Ihren Code in das Diff-Tool. Sollte es Unterschiede zwischen Ihrer Version des Codes und dem im Buch geben, werden sie Ihnen angezeigt.
- Schauen Sie im Web nach, ob jemand bereits die gleiche Frage gestellt (und eine Antwort darauf erhalten) hat.

Je besser Sie Ihre Programmierfragen formulieren, umso besser können andere Ihnen helfen. Gehen Sie daher in solchen Fällen immer wie folgt vor:

- Erklären Sie, was Sie zu tun versuchen, wenn der Fehler auftritt. Dadurch können potenzielle Helfer erkennen, ob Sie vielleicht völlig vom Wege abgekommen sind.
- Kopieren Sie die gesamte Fehlermeldung und Ihren Code.
- Geben Sie an, welches Betriebssystem und welche Version Sie benutzen.
- Beschreiben Sie, was Sie bereits getan haben, um das Problem zu lösen. Dadurch versichern Sie anderen, dass Sie schon selbst Anstrengungen unternommen haben, um der Sache auf den Grund zu gehen.
- Bleiben Sie höflich. Fordern Sie keine Hilfe und drängen Sie Ihre Helfer nicht, schnell zu antworten.

Nun sind Sie bereit zu lernen, wie Sie Ihre eigenen Computerspiele programmieren!

1

Die interaktive Shell

Bevor Sie Spiele entwickeln können, müssen Sie erst einige elementare Programmierprinzipien lernen. In diesem Kapitel erfahren Sie daher, wie Sie die interaktive Shell von Python nutzen und grundlegende Rechenoperationen durchführen.

Themen in diesem Kapitel

- Operatoren
- Integer und Fließkommazahlen
- Werte
- Ausdrücke
- Syntaxfehler
- Werte in Variablen speichern

Einfaches Rechnen

Starten Sie IDLE nach der Anleitung im Abschnitt »IDLE starten« in der Einleitung. Als Erstes wollen wir Python nutzen, um einige einfache Rechenaufgaben zu lösen. Die interaktive Shell lässt sich wie ein Taschenrechner benutzen. Geben Sie **2 + 2** an der Eingabeaufforderung >>> der Shell ein und drücken Sie die Eingabetaste (die je nach Tastatur [Enter] oder [Return] heißen kann). Abbildung 1–1 zeigt, wie die Rechenaufgabe in der interaktiven Shell angezeigt wird. Wie Sie sehen, antwortet die Shell mit der Zahl 4.

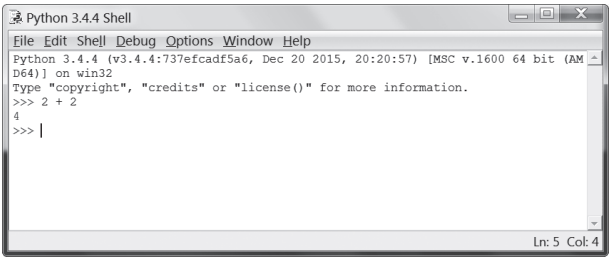


Abb. 1–1 *Eingabe von 2 + 2 in die aktive Shell*

Diese Rechenaufgabe ist ein einfacher Programmbefehl. Das Pluszeichen weist den Computer an, die beiden Zahlen 2 und 2 zu addieren. Der Computer erledigt diese Aufgabe und gibt in der folgenden Zeile die Zahl 4 aus. Tabelle 1–1 nennt weitere mathematische Symbole, die in Python zur Verfügung stehen.

Operator	Operation
+	Addition
–	Subtraktion
*	Multiplikation
/	Division

Tab. 1–1 *Mathematische Operatoren*

Das Minuszeichen subtrahiert Zahlen, das Sternchen multipliziert sie und der Schrägstrich teilt sie. Bei der Verwendung auf diese Weise werden +, –, * und / als *Operatoren* bezeichnet. Sie weisen Python an, was es mit den Zahlen auf beiden Seiten tun soll.

Integer und Fließkommazahlen

Integer sind ganze Zahlen wie 4, 99 oder 0. Bei *Fließkommazahlen* dagegen handelt es sich um Dezimalbrüche wie 3.5, 42.1 oder 5.0 (wobei in Programmiersprachen statt des im Deutschen üblichen Kommas ein Punkt verwendet wird). In Python ist 5 ein Integer und 5.0 eine Fließkommazahl.

Diese Zahlen werden als *Werte* bezeichnet. (Wir werden später noch andere Arten von Werten kennenlernen.) In der Rechenaufgabe, die Sie in die Shell eingegeben haben, sind 2 und 2 Integerwerte.

Ausdrücke

Die Rechenaufgabe 2 + 2 ist ein Beispiel für einen *Ausdruck*. Wie Abbildung 1–2 zeigt, bestehen Ausdrücke aus Werten (den Zahlen), die durch Operatoren (die Rechenzeichen) verbunden sind, um daraus einen neuen Wert zu bilden, der im Code verwendet werden kann. Computer können Millionen von Ausdrücken in Sekundenschnelle berechnen.

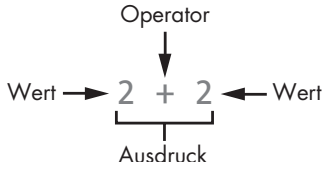


Abb. 1–2 *Ausdrücke bestehen aus Werten und Operatoren*

Versuchen Sie, einige Ausdrücke in die interaktive Shell einzugeben. Drücken Sie nach jedem dieser Ausdrücke Enter:

```
>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2 +         2
4
```

Diese Ausdrücke sehen alle wie ganz normale Rechenaufgaben aus. Beachten Sie aber die Leerzeichen in dem Beispiel `2 +  2`. In Python können Sie beliebig viele Leerzeichen zwischen Werte und Operatoren schreiben. Eingaben in die aktive Shell müssen jedoch immer am Anfang der Zeile beginnen (ohne Leerzeichen davor).

Ausdrücke auswerten

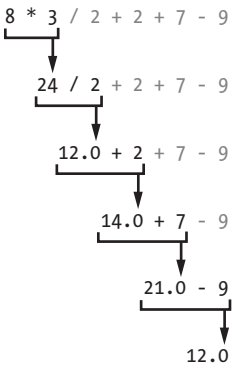
Wenn ein Computer den Ausdruck `10 + 5` verarbeitet und den Wert `15` zurückgibt, so spricht man davon, dass er den Ausdruck *auswertet*. Das bedeutet, dass der Ausdruck auf einen einzigen Wert reduziert wird. Wenn Sie eine Rechenaufgabe lösen, reduzieren Sie sie damit auch auf einen einzigen Wert, nämlich das Ergebnis. Die Ausdrücke `10 + 5` und `10 + 3 + 2` werden beide zu `15` ausgewertet.

Wenn Python einen Ausdruck auswertet, führt es die Operationen in der gleichen Reihenfolge aus, wie Sie es beim Rechnen tun. Dabei gelten die folgenden Regeln:

- Teile des Ausdrucks, die in Klammern stehen, werden zuerst ausgewertet.
- Multiplikation und Division erfolgen vor Addition und Subtraktion.
- Die Auswertung erfolgt von links nach rechts.

Der Ausdruck `1 + 2 * 3 + 4` ergibt daher `11` und nicht `13`, da zuerst `2 * 3` berechnet wird. Lautete die Aufgabe stattdessen `(1 + 2) * (3 + 4)`, würde sich `21` ergeben, da `(1 + 2)` und `(3 + 4)` in Klammern sehen und daher vor der Multiplikation ausgewertet werden.

Ausdrücke können eine beliebige Größe aufweisen, werden aber stets zu einem einzigen Wert ausgewertet. Auch einzelne Werte können Ausdrücke sein. So wird beispielsweise der Ausdruck `15` zu dem Wert `15` ausgewertet. Der Ausdruck `8 * 3 / 2 + 2 + 7 - 9` ergibt `21`. Dabei erfolgt die Berechnung in den folgenden Schritten:



Der Computer führt zwar alle diese Schritte durch, doch in der interaktiven Shell sehen Sie nichts davon, sondern nur das Ergebnis:

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

Ausdrücke mit dem Divisionsoperator `/` werden stets zu einer Fließkommazahl ausgewertet, hier beispielsweise `24 / 2` zu `12.0`. Mathematische Operationen, an

denen auch nur ein Fließkommawert beteiligt ist, ergeben wiederum Fließkommawerte. In unserem Beispiel wird deshalb `12.0 + 2` zu `14.0` ausgewertet.

Syntaxfehler

Wenn Sie `5 +` in die interaktive Shell eingeben, erhalten Sie die folgende Fehlermeldung:

```
>>> 5 +
SyntaxError: invalid syntax
```

Das liegt daran, dass `5 +` kein Ausdruck ist. Ausdrücke bestehen aus Werten, die durch Operatoren verbunden sind, und der Operator `+` erwartet einen Wert davor *und* dahinter. Wenn ein erwarteter Wert fehlt, wird eine Fehlermeldung angezeigt.

`SyntaxError` bedeutet, dass Python den Befehl nicht versteht, da Sie ihn falsch eingegeben haben. Bei der Programmierung geht es nicht einfach darum, dem Computer Befehle zu erteilen; man muss auch schon genau wissen, wie man diese Befehle auf die richtige Weise gibt!

Machen Sie sich jedoch keine Sorgen über solche Fehler. Durch diese Art falscher Angaben können Sie den Computer nicht beschädigen. Geben Sie den Befehl einfach an der nächsten Eingabeaufforderung (`>>>`) korrekt in die Shell ein.

Werte in Variablen speichern

Wenn ein Ausdruck zu einem Wert ausgewertet wird, können Sie diesen Wert später nutzen, wenn Sie ihn in einer *Variablen* speichern. Eine solche Variable können Sie sich als eine Schachtel vorstellen, in der Sie den Wert ablegen.

Eine *Zuweisungsanweisung* speichert einen Wert in einer Variablen. Geben Sie dazu den Namen der Variablen ein, dann ein Gleichheitszeichen – den sogenannten *Zuweisungsoperator* – und schließlich den Wert, den Sie speichern wollen:

```
>>> spam = 15
>>>
```

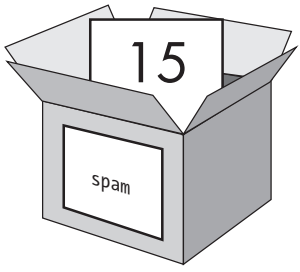


Abb. 1–3 *Variablen fungieren wie Schachteln, in denen Sie Werte aufbewahren können.*

Wenn Sie Enter drücken, scheint nichts zu passieren. In Python wissen Sie, dass ein Befehl erfolgreich ausgeführt wurde, wenn keine Fehlermeldung erscheint. Es wird einfach wieder die Eingabeaufforderung >>> angezeigt, sodass Sie den nächsten Befehl eingeben können.

Im Gegensatz zu Ausdrücken werden *Anweisungen* nicht zu einem Wert ausgewertet. Das ist der Grund, aus dem in der nächsten Zeile der interaktiven Shell nach `spam = 15` nichts mehr angezeigt wird. Wenn Ihnen nicht ganz klar ist, ob es sich bei einer Ihrer Eingaben um einen Ausdruck oder eine Anweisung handelt, so denken Sie daran, dass Ausdrücke immer zu einem einzelnen Wert ausgewertet werden. Alles andere sind Anweisungen.

In Variablen können Sie Werte festhalten, aber keine Ausdrücke. Betrachten Sie zum Beispiel die Ausdrücke in den Anweisungen `spam = 10 + 5` und `spam = 10 + 7 - 2`. Beide Ausdrücke werden zu `15` ausgewertet, und beide Anweisungen speichern den Wert `15` in der Variablen `spam`.

Die Beispielvariablen in diesem Kapitel tragen nichtssagende Namen wie `spam`, `eggs` und `bacon`, doch für die Variablen in Ihren Programmen sollten Sie Namen verwenden, die die enthaltenen Daten beschreiben. Stellen Sie sich vor, Sie würden bei einem Umzug alle Kartons einfach mit *Sachen* beschriften. Sie würden nichts mehr wiederfinden!

Python erstellt die Variable, wenn sie zum ersten Mal in einer Zuweisungsanweisung verwendet wird. Um zu prüfen, ob sich ein Wert in der Variablen befindet, geben Sie den Variablennamen in die interaktive Shell ein:

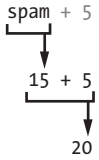
```
>>> spam = 15
>>> spam
15
```

Der Ausdruck `spam` wird zu dem Wert ausgewertet, der sich in der Variablen `spam` befindet, also `15`.

Variablen können Sie auch in Ausdrücken verwenden. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = 15
>>> spam + 5
20
```

Hier haben Sie zunächst den Wert der Variablen `spam` auf `15` gesetzt, sodass die Eingabe `spam + 5` dem Ausdruck `15 + 5` entspricht. Die Auswertung von `spam + 5` läuft wie folgt ab:



Es ist nicht möglich, eine Variable schon vor der Zuweisungsanweisung zu verwenden, mit der sie erstellt wird. Sollten Sie das versuchen, so gibt Python die Fehlermeldung `NameError` aus, da eine Variable des angegebenen Namens noch nicht existiert. Der gleiche Fehler ergibt sich auch, wenn Sie den Variablennamen falsch schreiben:

```
>>> spam = 15
>>> spma
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    spma
NameError: name 'spma' is not defined
```

Der Fehler tritt auf, da es nur eine Variable namens `spam` gibt, aber nicht `spma`.

Den Wert, der in einer Variablen gespeichert ist, können Sie mit einer weiteren Zuweisungsanweisung ändern. Probieren Sie das folgende Beispiel in der interaktiven Shell aus:

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

Wenn Sie zum ersten Mal `spam + 5` eingeben, wird der Ausdruck zu `20` ausgewertet, da in `spam` der Wert `15` gespeichert ist. Dann allerdings geben Sie `spam = 3` ein, wodurch der Wert `15` in der »Schachtel« der Variablen gegen den Wert `3` ausgetauscht oder mit ihm *überschrieben* wird, da eine Variable immer nur einen Wert auf einmal enthalten kann. Der Wert von `spam` lautet jetzt `3`, weshalb der Ausdruck `spam + 5` bei der erneuten Eingabe zu `8` ausgewertet wird. Beim Überschreiben wird praktisch der vorherige Wert aus der Variablenschachtel herausgenommen und der neue hineingelegt, wie Abbildung 1–4 illustriert.

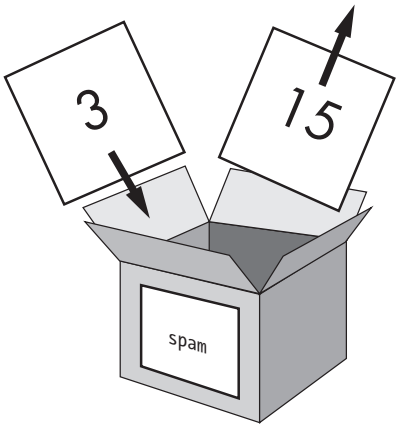


Abb. 1–4 *Der Wert* 15 *in* spam *wird durch den Wert* 3 *überschrieben.*

Sie können den Wert in spam sogar dazu verwenden, spam selbst einen neuen Wert zuzuweisen:

```
>>> spam = 15
>>> spam = spam + 5
>>> spam
20
```

Die Zuweisungsanweisung `spam = spam + 5` bedeutet: »Der neue Wert der Variablen spam ist der alte Wert von spam plus 5.« Um den Wert in spam wiederholt um 5 zu erhöhen, geben Sie Folgendes in die interaktive Shell ein:

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

In der ersten Anweisung dieses Beispiels speichern Sie den Wert 15 in spam. In der nächsten addieren Sie 5 zu dem Wert von spam und weisen den neuen Wert `spam + 5`, der 20 ergibt, der Variablen spam zu. Wenn Sie das dreimal tun, enthält spam schließlich den Wert 30.

Bis jetzt haben wir uns nur mit einer einzigen Variablen beschäftigt, doch Sie können in Ihren Programmen so viele Variablen verwenden, wie Sie benötigen. Als Beispiel wollen wir den beiden Variablen eggs und bacon unterschiedliche Werte zuweisen:

```
>>> bacon = 10
>>> eggs = 15
```

Die Variable bacon enthält jetzt den Wert 10, während sich in eggs der Wert 15 befindet. Jede Variable stellt eine eigene Schachtel mit einem eigenen Wert dar, wie Sie in Abbildung 1–5 sehen.

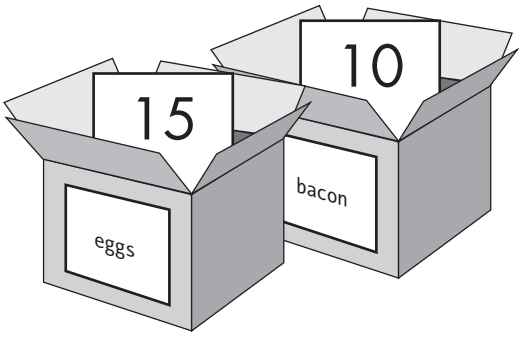


Abb. 1–5 *Unterschiedliche Werte in den Variablen* bacon *und* eggs

Geben Sie jetzt spam = bacon + eggs in die interaktive Shell ein und prüfen Sie dann den Wert von spam:

```
>>> bacon = 10
>>> eggs = 15
>>> spam = bacon + eggs
>>> spam
25
```

Der Wert in spam beträgt jetzt 25. Bei der Addition von bacon und eggs werden deren Werte addiert, also 10 und 15. Da Variablen Werte enthalten und keine Ausdrücke, wurde spam der Wert 25 und nicht der Ausdruck bacon + eggs zugewiesen. Nachdem Sie mit der Anweisung spam = bacon + eggs den Wert 25 zu spam zugewiesen haben, haben anschließende Änderungen von bacon oder eggs keinen Einfluss auf spam.

Zusammenfassung

In diesem Kapitel haben Sie die Grundlagen zum Schreiben von Python-Programmbefehlen gelernt. Da Computer nicht denken können und nur bestimmte Befehle verstehen, müssen Sie Python genau sagen, was es tun soll.

Ausdrücke sind Werte (wie 2 oder 5), die durch Operatoren (wie + und -) verbunden sind. Python kann Ausdrücke auswerten, wobei sie auf einen einzigen Wert reduziert werden. Werte können Sie auch in Variablen speichern, sodass das Programm sich diese Werte merkt und später verwenden kann.