Ein Buch zum Mitmachen und Verstehen

Programmieren

von Kopf bis Fuß

Laden Sie sich die wichtigen Konzepte direkt in Ihr Hirn

Bestücken Sie Ihren Werkzeugkasten mit Methoden, Funktionen und Objekten

Vermeiden Sie nervige Ein-/Ausgabe-Pannen

Eine allgemeine Einführung, deren Beispiele in Python geschrieben sind

Verarbeiten Sie Ihre Daten wie ein echter Profi

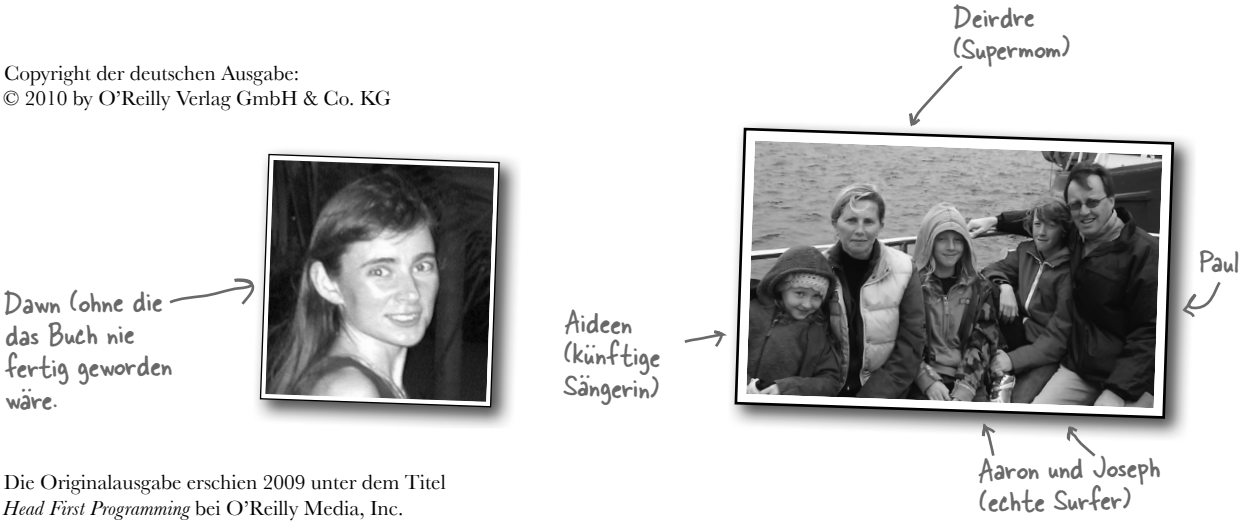
Bauen Sie so funktionelle wie attraktive grafische Anwendungen

Erfahren Sie, wie sich wiederkehrende Aufgaben automatisieren lassen

O'REILLY®

Paul Barry & David Griffiths
Deutsche Übersetzung von Lars Schulten

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
E-Mail: kommentar@oreilly.de



Die Originalausgabe erschien 2009 unter dem Titel
Head First Programming bei O'Reilly Media, Inc.

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.

Deutsche Übersetzung und Bearbeitung: Jörg Beyer, Weimar (Lahn)
Lektorat: Christine Haite, Köln
Korrektorat: Sibylle Feldmann, Düsseldorf
Satz: Ulrich Borstelmann, Dortmund
Umschlaggestaltung: Karen Montgomery, Boston
Produktion: Andrea Miß, Köln
Belichtung, Druck und buchbinderische Verarbeitung: Media-Print, Paderborn

ISBN 978-3-89721-992-2

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Bei der Produktion dieses Buchs kamen keine Daten zu Schaden.

Wir widmen dieses Buch dem Ersten, der vor einem Computer stand und sich die Frage stellte: »Wie *bringe* ich ihn dazu, das zu tun ... ?«

Und denen, die das Programmieren so komplex machten, dass man ein Buch wie dieses braucht, um es zu lernen.

David: Für Dawn, den klügsten Menschen, der mir je begegnet ist.

Paul: Dieses Buch ist meinem Vater gewidmet, der mich vor 25 Jahren – als ich einen Anstoß brauchte – zur Computerei trieb. Das war ein guter Schubs.

Die Autoren von Programmieren von Kopf bis Fuß

Paul Barry hat vor Kurzem festgestellt, dass er seit fast einem Vierteljahrhundert programmiert. Was ihn nicht wenig schockierte. In dieser Zeit hat er mit vielen verschiedenen Programmiersprachen gearbeitet, in zwei Ländern auf zwei Kontinenten gelebt und gewirkt, geheiratet, drei Kinder bekommen (okay, *bekommen* hat sie eigentlich seine Frau Deirdre, aber irgendwie hat er ja auch seinen Teil geleistet), einen B.Sc. und einen M.Sc. in Computing erworben, neben zwei weiteren Büchern einen Haufen technischer Artikel für das *Linux Journal* geschrieben und es irgendwie geschafft, seine Haare *nicht* zu verlieren ... was sich leider bald ändern könnte.

Als er das erste Mal voller Begeisterung einen Blick auf *HTML mit CSS & XHTML von Kopf bis Fuß* warf, erkannte er sofort, dass dieses Konzept genau der richtige Ansatz war, um Programmieren zu lehren. Er war daher nur zu erfreut, als er gemeinsam mit David die Möglichkeit erhielt, zu zeigen, dass diese Ahnung richtig war.

Normalerweise arbeitet Paul als Lehrer am Institute of Technology, Carlow, in Irland. Als Mitglied der Abteilung für »Computing & Networking« verbringt er seine Tage damit, coole Programmiertechnologien zu erforschen und zu lehren und damit genau das zu tun, was er für den besten Zeitvertreib hält. Seine große Hoffnung ist, dass seine Studenten den Kram, den er sie lehrt, ebenso amüsant finden.

David Griffiths begann im zarten Alter von 12 mit dem Programmieren, nachdem er eine Dokumentation über die Arbeiten von Seymour Papert gesehen hatte. Mit 15 schrieb er eine Implementierung von Paperts Programmiersprache LOGO. Nachdem er an der Universität Mathematik studiert hatte, begann er, Code für Computer und Zeitschriftenartikel für Menschen zu schreiben. Er arbeitete als Trainer für agile Softwareentwicklung, Entwickler und Parkplatzwächter, allerdings nicht in dieser Reihenfolge. Er kann in über zehn Sprachen programmieren, schreiben aber nur in einer. Wenn er nicht schreibt, programmiert oder lehrt, verbringt er seine Zeit auf Reisen mit seiner Frau – und *Von Kopf bis Fuß*-Autorenkollegin – Dawn.

Bevor er *Programmieren von Kopf bis Fuß* schrieb, schrieb er ein anderes Buch namens *Head First Rails*, das ebenfalls sehr lesenswert und ein ausgezeichnetes Geschenk für einen guten Freund oder Verwandten ist.

Sie können ihm auf Twitter folgen:

```
http://twitter.com/dgriffiths
```

Über den Übersetzer dieses Buchs

Lars Schulten ist freier Übersetzer für IT-Fachliteratur und hat für den O'Reilly Verlag schon unzählige Bücher zu ungefähr allem übersetzt, was man mit Computern so anstellen kann. Eigentlich hat er mal Philosophie studiert, aber mit Computern schlägt er sich schon seit den Zeiten herum, da Windows laufen lernte. Die Liste der Dinge, mit denen er sich beschäftigt, ist ungefähr so lang, launenhaft und heterogen wie die seiner Lieblingsessen und Lieblingsbücher.

Allein tritt er eigentlich nur auf, wenn er mal wieder versucht, den körperlichen Verfall mit sportlicher Betätigung aufzuhalten. Sonst ist er immer in Begleitung eines Buchs, seines Laptops oder Frederics unterwegs. Frederic ist sechs Jahre alt und setzt gern eine sehr kritische Miene auf, wenn Papa die Spielerei mit dem Computer als Arbeit bezeichnet.

Verwandte Bücher von O'Reilly

Einführung in Python

Programmieren mit Python

Python Cookbook

Weitere Bücher aus O'Reillys *Von Kopf bis Fuß*-Reihe

Java™ von Kopf bis Fuß

Objektorientierte Analyse und Design (OOA&D) von Kopf bis Fuß

Softwareentwicklung von Kopf bis Fuß

Entwurfsmuster von Kopf bis Fuß

Servlets und JSP von Kopf bis Fuß

C# von Kopf bis Fuß

SQL von Kopf bis Fuß

PHP & MySQL von Kopf bis Fuß

HTML mit CSS und XHTML von Kopf bis Fuß

JavaScript von Kopf bis Fuß

Datenanalyse von Kopf bis Fuß

Statistik von Kopf bis Fuß

Head First Algebra

Head First Physics

Head First Ajax

Head First Rails

Webdesign von Kopf bis Fuß

Head First PMP

Head First EJB

Der Inhalt (im Überblick)

Der Inhalt (jetzt ausführlich)

Einführung

Ihr Gehirn und das Programmieren. *Sie* versuchen, etwas zu *lernen*, und Ihr *Hirn* tut sein Bestes, damit das Gelernte nicht *hängen bleibt*. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z.B. für das Wissen, welche Tiere einem gefährlich werden könnten, oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, *wie* schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, etwas über das Programmieren zu wissen?

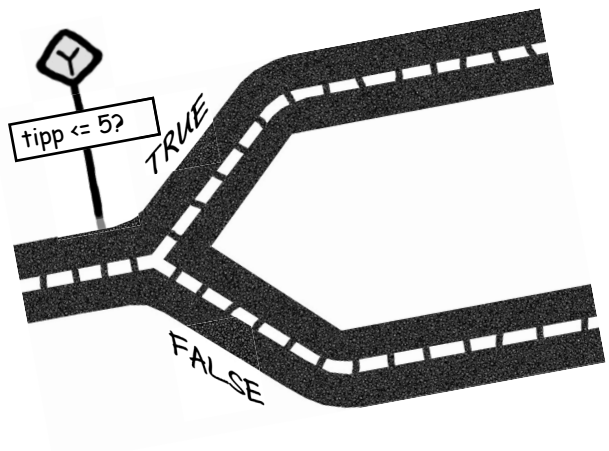
Der Anfang des Programmierens

1 Orientierungshilfe

Eigene Programme verleihen Ihnen die Macht, Ihren PC zu steuern.

Fast jeder weiß, wie man mit einem Computer *arbeitet*, aber nur wenige machen den nächsten Schritt und lernen, wie man ihn *steuert*. Wenn Sie Software nutzen, die von anderen entwickelt wurde, sind Sie immer auf das beschränkt, was Sie *deren* Meinung nach tun möchten. Schreiben Sie Ihre eigenen Programme, und nur noch Ihre eigene Vorstellungskraft kann Sie einschränken. Das Programmieren macht Sie kreativer, lässt Sie präziser denken und lehrt Sie, Probleme logisch zu analysieren und zu lösen.

Wollen Sie lieber programmiert oder Programmierer sein?

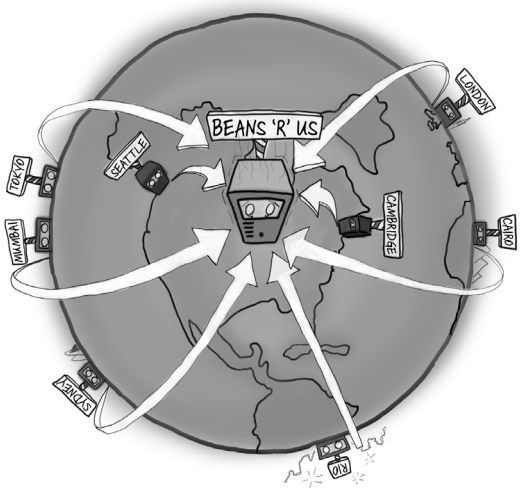


Textdaten

2 Jeder Text zu seiner Zeit

Stellen Sie sich vor, Sie müssten ohne Worte kommunizieren.

Alle Programme verarbeiten Daten – und eine der wichtigsten Arten von Daten ist **Text**. In diesem Kapitel werden Sie das Basiswissen zu **Textdaten** erhalten. Sie werden Text **suchen** und automatisch genau das bekommen, **was Sie suchten**. Und unterwegs werden Sie grundlegende Konzepte der Programmierung wie **Methoden** aufsammeln und erfahren, wie Sie sie einsetzen können, **um sich Ihre Daten gefügig zu machen**. Und schließlich werden Sie Ihren Programmen mithilfe von **Codebibliotheken** im Handumdrehen **Superkräfte geben**.



Funktionen

3 Ordnung schaffen

Wenn Programme wachsen, wird Code unübersichtlich.

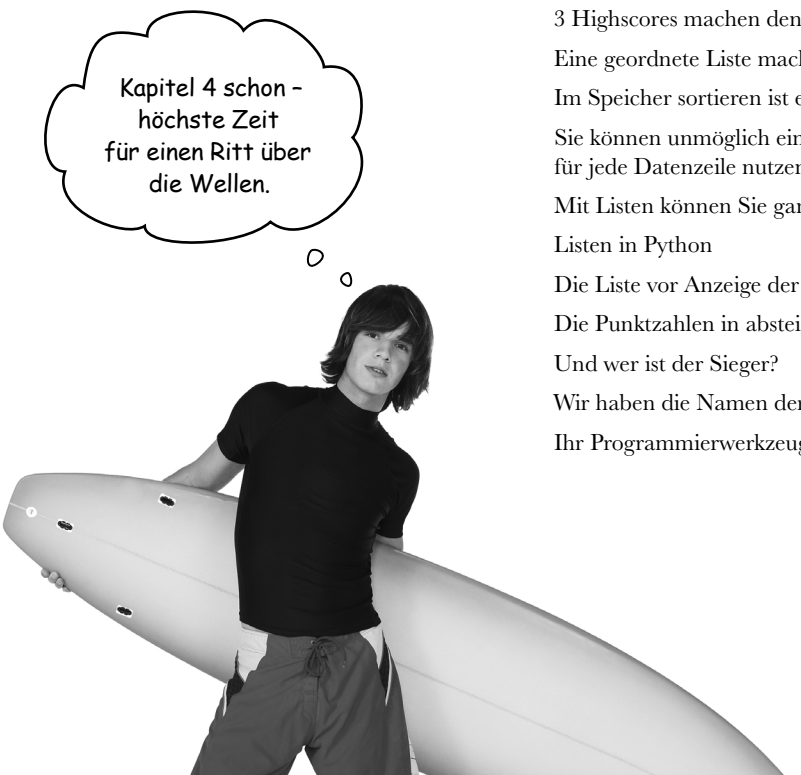
Unübersichtlicher und komplexer Code kann schwer zu lesen und noch schwerer zu warten sein. Eine Möglichkeit, diese Komplexität in den Griff zu bekommen, ist der Einsatz von **Funktionen**. Funktionen sind **Codeeinheiten**, die Sie nach Bedarf in Ihren Programmen einsetzen. Sie ermöglichen Ihnen, **gemeinsam genutzte Aktionen auszulagern**, und das bedeutet, dass Sie Ihren Code **besser lesbar** und **leichter wartbar** machen. In diesem Kapitel werden Sie entdecken, wie Ihnen etwas Wissen zu Funktionen **das Programmiererleben erheblich vereinfachen kann**.

Daten in Dateien und Listen

4 Sortieren

Mit Ihren Programmen entwickeln sich auch die Datenverarbeitungsanforderungen.

Und wenn Sie mit vielen Daten arbeiten müssen, wird es schnell recht mühselig, einzelne Variablen für jedes einzelne Datenelement zu verwenden. Programmierer nutzen deswegen ziemlich beeindruckende Behälter oder Container (die als **Datenstrukturen** bezeichnet werden), um sich die Arbeit mit vielen Daten zu vereinfachen. Häufig kommen alle Daten aus Dateien, die auf der Festplatte gespeichert sind. Aber wie arbeitet man mit Daten in Dateien? Es wird sich herausstellen, dass es ein Kinderspiel ist. Blättern Sie um und erfahren Sie, warum!



Abbildungen und Datenbanken

5 Daten ihren Platz zuweisen

Listen sind nicht der einzige Datencontainer.

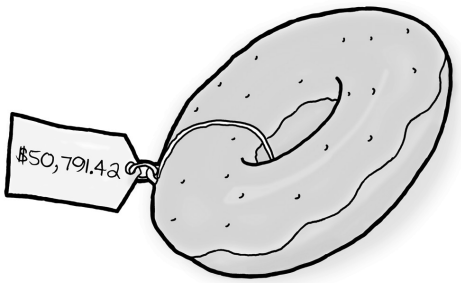
Programmiersprachen bieten weitere Spielzeuge zum Festhalten von Daten, und unser erwähltes Werkzeug, Python, macht da keine Ausnahme. In diesem Kapitel werden Sie Werte mit Namen **verknüpfen** und dazu eine Datenstruktur nutzen, die unter vielen Namen bekannt ist, Hash, Tabelle, **Abbildung**, und unter Python *Dictionary*. Und bei der Speicherung von Daten wollen wir uns jetzt nicht mehr auf Textdaten in Dateien beschränken, sondern auf Daten in einem *externen Datenbanksystem zugreifen*. **Information** ist alles, und da es die ohne Daten nicht gibt, blättern Sie um und beginnen damit, Ihre stetig wachsenden Programmierfertigkeiten auf einige coole Datenverarbeitungsaufgaben anzuwenden.

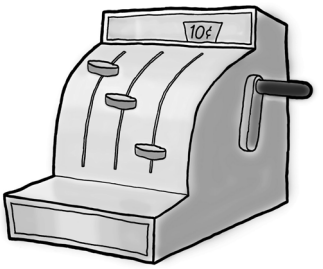
Modulare Programmierung

6 In der Spur bleiben

Ihr Code geht in viele Programme ein.

Und obgleich **Teilen** eine gute Sache ist, müssen Sie *aufpassen*. Es könnte sein, dass Programmierer Ihren Code nehmen und auf **unerwartete** Weise nutzen oder andere ihn einfach ändern, ohne Sie darüber zu informieren. Eventuell möchten Sie eine Funktion in all Ihren Programmen nutzen, und mit der Zeit **ändert** sich der Code dieser Funktion, damit er weiterhin Ihre Anforderungen erfüllt. Schlaue Programmierer nutzen *modulare Programmiertechniken*, um ihre Arbeitsbelastung unter Kontrolle zu halten. Finden Sie auf den folgenden Seiten heraus, wie man das anstellt ...





7 Grafische Benutzeroberflächen

Augen- und Ohrenschmaus

Ihre Fertigkeiten als Programmierer nehmen stetig zu.

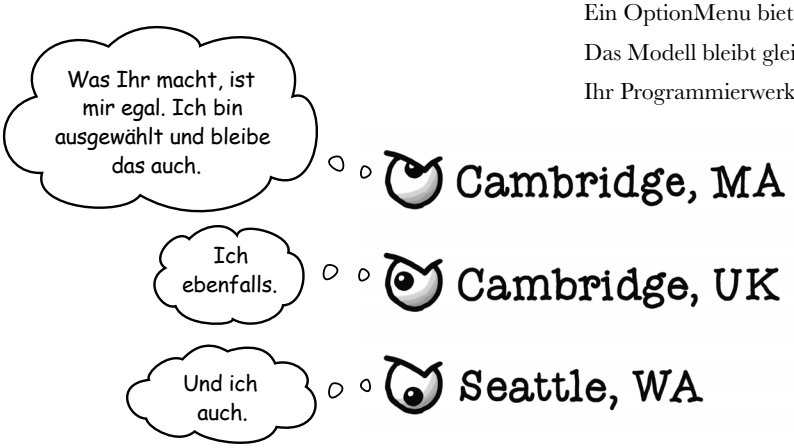
Aber es ist ein Jammer, dass Ihre Programme nicht *ansehnlicher* sind. Fragen und Antworten auf der Konsole anzeigen zu können, ist ja ganz hübsch, aber doch recht retro, oder nicht? Fehlt eigentlich nur noch grüner Text auf schwarzem Hintergrund zum wahren IT-Steinzeitgefühl. Es muss doch eine *bessere* Möglichkeiten geben, mit Benutzern zu kommunizieren – und die gibt es natürlich: **grafische Benutzeroberflächen** oder **GUIs**. Klingt cool und komplex und kann beides sein. Aber keine Angst, mit ein zwei Tricks haben Sie Ihren grafischen Code in null Komma nichts laufen.

GUIs und Daten

8 Grafische Dateneingabe

GUIs verarbeiten nicht nur Ereignisse, sondern auch Daten.

Fast alle GUI-Anwendungen müssen Benutzerdaten lesen, und die Wahl der richtigen Eingabeelemente kann entscheiden, ob Ihre Benutzeroberfläche eine *Dateneingabehölle* oder der siebte *Benutzerhimmel* ist. Widgets können einfachen Text akzeptieren oder nur eine Optionsliste präsentieren. Es gibt viele verschiedene Widgets, und das heißt, dass Sie viele Optionen haben. Welche Sie davon wählen, kann natürlich ganz entscheidende Auswirkungen haben. Es ist an der Zeit, dass Sie Ihre GUI-Programme **weiterentwickeln**.



8½

Ausnahmen und Dialogfenster

Nachricht angekommen?

Manchmal geht etwas schief, und Sie müssen sich darum kümmern.

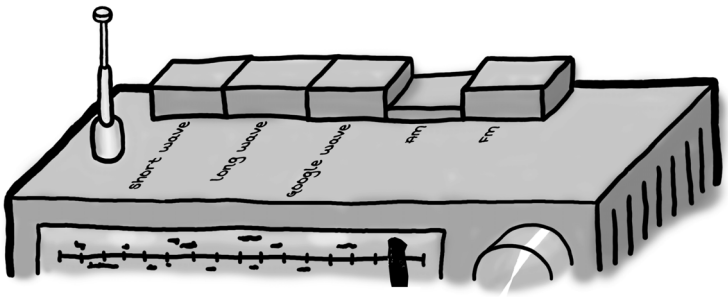
Es gibt immer Dinge, die Sie nicht im Griff haben. Netzwerke fallen aus. Dateien verschwinden. Schlaue Programmierer wissen, wie sie mit derartigen **Fehlern** umgehen, und bieten in ihren Programmen **Ausweichlösungen**. Gute Software informiert den Nutzer, wenn etwas Übles passiert, und sagt ihm, was erforderlich ist, um die Angelegenheit zu regeln. Wenn Sie lernen, wie Sie **Ausnahmen** und **Dialogfenster** einsetzen, können Sie die Zuverlässigkeit und Qualität Ihrer Software steigern.

Elemente grafischer Oberflächen

9 Das richtige Werkzeug wählen

Gut verwendbare Programme zu schreiben, ist nicht schwer..

Bei GUI-Anwendungen gibt es einen ganz entscheidenden Unterschied zwischen *funktionierenden* Schnittstellen und **nützlichen** sowie **effektiven** Schnittstellen. Wie man das richtige Werkzeug für eine Aufgabe wählt, lernt man durch Erfahrung, und die erwirbt man nur, indem man mit den verfügbaren Werkzeugen arbeitet. In diesem Kapitel werden Sie Ihre Fertigkeiten im Aufbau von GUI-Anwendungen erweitern. Es gibt noch eine Menge nützlicher Widgets, die auf Sie warten. Blättern Sie also um und lassen Sie uns fortfahren.

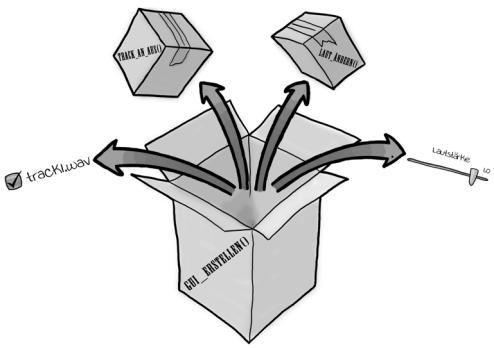


Eigene Widgets und Klassen

10 Objekte im Sinn

Anforderungen können komplex sein, Programme müssen es nicht sein.

Indem Sie Objektorientierung nutzen, können Sie Ihre Programme sehr **leistungsfähig** machen, ohne dazu Unmengen an Code zusätzlich schreiben zu müssen. Lesen Sie weiter und erfahren Sie, wie Sie **eigene Widgets** erstellen, die genau das machen, was Sie *wollen*, und Ihnen dazu verhelfen, **Ihre Programmierfertigkeiten weiterzuentwickeln**.

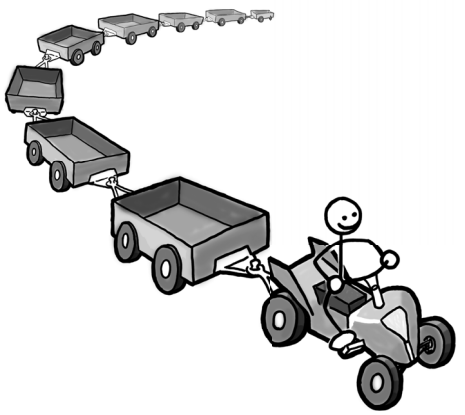


Anhang: Was übrig bleibt

Die Top-Ten der Dinge (die wir nicht behandelt haben)

Sie haben einen weiten Weg zurückgelegt.

Doch programmieren zu lernen ist eine Aufgabe, die nie abgeschlossen ist. Je mehr Sie programmieren, umso wichtiger wird es, dass Sie **neue Wege erforschen, um bestimmte Dinge zu tun**. Sie müssen sich mit **neuen Werkzeugen** und **neuen Techniken** vertraut machen. Leider bietet dieses Buch einfach nicht genügend Raum dafür, Ihnen all das zu zeigen, was eventuell nützlich für Sie werden könnte. Deswegen finden Sie hier eine Liste der zehn wichtigsten Dinge, die wir nicht behandelt haben, die Sie vielleicht aber als Nächstes lernen sollten.



Einführung

In diesem Abschnitt beantworten wir die brennende Frage: »Und? Warum STEHT so was in einer Einführung in die Programmierung?«

Für wen ist dieses Buch?

Wenn Sie alle folgenden Fragen mit »Ja« beantworten können:

1. Wünschten Sie, Sie wüssten, wie Sie Ihren Computer steuern und dazu bringen können, ganz neue Dinge zu tun?
2. Möchten Sie programmieren lernen, damit Sie der neue Stern am Softwarehimmel werden, Millionen scheffeln und sich bald auf Ihrer eigenen Insel zur Ruhe setzen können?

Gut, das ist vielleicht etwas übertrieben, aber alle haben ja mal klein angefangen, oder?

3. Ziehen Sie es vor, tatsächlich etwas zu tun? Die Dinge anzuwenden, die Sie gelernt haben? Ist das besser, als stundenlang jemandem zuzuhören, der sich in einem endlosen Vortrag über irgendwas ergeht?

... dann ist dieses Buch etwas für Sie.

Wer sollte eher die Finger von diesem Buch lassen?

Wenn Sie eine dieser Fragen mit »Ja« beantworten müssen:

1. Sind Sie ein erfahrener Programmierer? Wissen Sie bereits, wie man programmiert?
2. Suchen Sie nach einer knappen Einführung in oder eine Referenz für Python?
3. Würden Sie sich lieber von 15 kreischenden Affen die Zehennägel ziehen lassen, als einmal etwas Neues auszuprobieren? Sind Sie der Ansicht, dass ein Buch zur Programmierung *wirklich alles* behandeln muss und richtig gut eigentlich nur sein kann, wenn es den Leser zu Tode langweilt?

... dann ist dieses Buch ***nicht*** das richtige für Sie.

[Anmerkung aus dem Marketing: Dieses Buch ist etwas für jeden, der eine Kreditkarte besitzt. Und auch Barzahlung ist möglich.]

Wir wissen, was Sie gerade denken.

»Wie kann *das* ein ernsthaftes Buch zur Programmierung sein?«

»Was sollen all die Abbildungen?«

»Kann ich auf diese Weise wirklich *lernen*?«

Und wir wissen, was Ihr *Gehirn* gerade denkt.

Ihr Gehirn lechzt nach Neuem. Es ist ständig dabei, Ihre Umgebung abzusuchen, und es *wartet* auf etwas Ungewöhnliches. So ist es nun einmal gebaut, und es hilft Ihnen zu überleben.

Also, was macht Ihr Gehirn mit all den gewöhnlichen, normalen Routinesachen, denen Sie begegnen? Es tut alles in seiner Macht stehende, damit es dadurch nicht bei seiner *eigentlichen* Arbeit gestört wird: Dinge zu erfassen, die wirklich *wichtig* sind. Es gibt sich nicht damit ab, die langweiligen Sachen zu speichern, sondern lässt sie gar nicht erst durch den »Dies-ist-offensichtlich-nicht-wichtig«-Filter.

Woher *weiß* Ihr Gehirn denn, was wichtig ist? Nehmen Sie an, Sie machen einen Tagesausflug und ein Tiger springt vor Ihnen aus dem Gebüsch: Was passiert dabei in Ihrem Kopf und Ihrem Körper?

Neuronen feuern. Gefühle werden angekurbelt. *Chemische Substanzen durchfluten Sie.*

Und so weiß Ihr Gehirn:

Dies muss wichtig sein! Vergiss es nicht!

Aber nun stellen Sie sich vor, Sie sind zu Hause oder in einer Bibliothek. In einer sicheren, warmen, tigerfreien Zone. Sie lernen. Bereiten sich auf eine Prüfung vor. Oder Sie versuchen, irgendein schwieriges Thema zu lernen, von dem Ihr Chef glaubt, Sie bräuchten dafür eine Woche oder höchstens zehn Tage.

Da ist nur ein Problem: Ihr Gehirn versucht, Ihnen einen großen Gefallen zu tun. Es versucht, dafür zu sorgen, dass diese *offensichtlich* unwichtigen Inhalte nicht knappe Ressourcen verstopfen. Ressourcen, die besser dafür verwendet würden, die wirklich *wichtigen* Dinge zu speichern. Wie Tiger. Wie die Gefahren des Feuers. Die späte Einsicht, dass Sie nie diese »Partybilder« auf Ihre Facebook-Seite hätten stellen dürfen. Und es gibt keine einfache Möglichkeit, Ihrem Gehirn zu sagen: »Hey, Gehirn, vielen Dank, aber egal, wie langweilig dieses Buch auch ist und wie klein der Ausschlag auf meiner emotionalen Richterskala gerade ist, ich *will* wirklich, dass du diesen Kram behältst.«

Wir stellen uns unseren Leser als einen aktiv Lernenden vor.

Also, was ist nötig, damit Sie etwas *lernen*? Erst einmal müssen Sie es *aufnehmen* und dann dafür sorgen, dass Sie es nicht wieder *vergessen*. Es geht nicht darum, Fakten in Ihren Kopf zu schieben. Nach den neuesten Forschungsergebnissen der Kognitionswissenschaft, der Neurobiologie und der Lernpsychologie gehört zum *Lernen* viel mehr als nur Text auf einer Seite. Wir wissen, was Ihr Gehirn anmacht.

Einige der Lernprinzipien dieser Buchreihe:

Wir setzen Bilder ein. An Bilder kann man sich viel besser erinnern als an Worte allein und lernt so viel effektiver (bis zu 89% Verbesserung bei Abrufbarkeits- und Lerntransferstudien). Außerdem werden die Dinge dadurch verständlicher. **Wir setzen Text in oder neben die Grafiken,** auf die sie sich beziehen, anstatt darunter oder auf eine andere Seite. Die Leser werden auf den Bildinhalt bezogene Probleme dann mit *doppelt* so hoher Wahrscheinlichkeit lösen können.

Wir verwenden einen gesprächsorientierten Stil mit persönlicher Ansprache. Nach neueren Untersuchungen haben Studenten nach dem Lernen bei Tests bis zu 40% besser abgeschnitten, wenn der Inhalt den Leser direkt in der ersten Person und im lockeren Stil angesprochen hat statt in einem formalen Ton. Wir halten daher keinen Vortrag, sondern erzählen Geschichten. Wir benutzen eine zwanglose Sprache. Wir nehmen uns selbst nicht zu ernst. Würden *Sie* einer anregenden Unterhaltung beim Abendessen mehr Aufmerksamkeit schenken oder einem Vortrag?

Wir bringen den Lernenden dazu, intensiver nachzudenken. Mit anderen Worten: Falls Sie nicht aktiv Ihre Neuronen strapazieren, passiert in Ihrem Gehirn nicht viel. Ein Leser muss motiviert, begeistert und neugierig sein und angeregt werden, Probleme zu lösen, Schlüsse zu ziehen und sich neues Wissen anzueignen. Und dafür brauchen Sie Herausforderungen, Übungen, zum Nachdenken anregende Fragen und Tätigkeiten, die beide Seiten des Gehirns und mehrere Sinne einbeziehen.

Wir ziehen die Aufmerksamkeit des Lesers auf uns – und behalten sie. Wir alle haben schon Erfahrungen dieser Art gemacht: »Ich will das wirklich lernen, aber ich kann einfach nicht über Seite 1 hinaus wach bleiben.« Ihr Gehirn passt auf, wenn Dinge ungewöhnlich, interessant, merkwürdig, auffällig, unerwartet sind. Ein neues, schwieriges, technisches Thema zu lernen, muss nicht langweilig sein. Wenn es das nicht ist, lernt Ihr Gehirn viel schneller.

Wir sprechen Gefühle an. Wir wissen, dass Ihre Fähigkeit, sich an etwas zu erinnern, wesentlich von dessen emotionalem Gehalt abhängt. Sie erinnern sich an das, was Sie *bewegt*. Sie erinnern sich, wenn Sie etwas *fühlen*. Nein, wir erzählen keine herzzerreißenden Geschichten über einen Jungen und seinen Hund. Was wir erzählen, ruft Überraschungs-, Neugier-, Spaß- und Was-soll-das?-Emotionen hervor und dieses Hochgefühl, das Sie beim Lösen eines Puzzles empfinden oder wenn Sie etwas lernen, was alle anderen schwierig finden. Oder wenn Sie merken, dass Sie etwas können, das dieser »Ich-bin-ein-besserer-Techniker-als-du«-Typ aus der Technikabteilung *nicht kann*.

Metakognition: Nachdenken übers Denken

Wenn Sie wirklich lernen möchten, und zwar schneller und nachhaltiger, dann schenken Sie Ihrer Aufmerksamkeit Aufmerksamkeit. Denken Sie darüber nach, wie Sie denken. Lernen Sie, wie Sie lernen.

Die meisten von uns haben in ihrer Jugend keine Kurse in Metakognition oder Lerntheorie gehabt. Es wurde von uns *erwartet*, dass wir lernen, aber nur selten wurde uns auch *beigebracht*, wie man lernt.

Wir nehmen aber an, dass Sie wirklich lernen möchten, wie man programmiert, wenn Sie dieses Buch in den Händen halten. Und wahrscheinlich möchten Sie nicht viel Zeit aufwenden. Und Sie wollen sich an das *erinnern*, was Sie lesen, und es anwenden können. Und deshalb müssen Sie es *verstehen*. Wenn Sie so viel wie möglich von diesem Buch profitieren wollen oder von *irgendeinem* anderen Buch oder einer anderen Lernerfahrung, übernehmen Sie Verantwortung für Ihr Gehirn. Ihr Gehirn im Zusammenhang mit *diesem* Lernstoff.

Der Trick besteht darin, Ihr Gehirn dazu zu bringen, neuen Lernstoff als etwas wirklich Wichtiges anzusehen. Als entscheidend für Ihr Wohlbefinden. So wichtig wie ein Tiger. Andernfalls stecken Sie in einem dauernden Kampf, in dem Ihr Gehirn sein Bestes gibt, um die neuen Inhalte davon abzuhalten, hängen zu bleiben.

Wie bringen Sie also Ihr Gehirn dazu, das Programmieren für so wichtig zu halten wie einen Tiger?

Da gibt es den langsamen, ermüdenden Weg oder den schnelleren, effektiveren Weg. Der langsame Weg geht über bloße Wiederholung. Natürlich ist Ihnen klar, dass Sie lernen und sich sogar an die langweiligsten Themen erinnern *können*, wenn Sie sich die gleiche Sache immer wieder einhämmern. Wenn Sie etwas nur oft genug wiederholen, sagt Ihr Gehirn: »Er hat zwar nicht das *Gefühl*, dass das wichtig ist, aber er sieht sich dieselbe Sache *immer und immer wieder* an – dann muss sie wohl wichtig sein.«

Der schnellere Weg besteht darin, ***alles zu tun, was die Gehirnaktivität erhöht***, vor allem verschiedene *Arten* von Gehirnaktivität. Eine wichtige Rolle dabei spielen die auf der vorhergehenden Seite erwähnten Dinge – alles Dinge, die nachweislich helfen, dass Ihr Gehirn *für* Sie arbeitet. So hat sich z.B. in Untersuchungen gezeigt: Wenn Wörter *in* den Abbildungen stehen, die sie beschreiben (und nicht irgendwo anders auf der Seite, z.B. in einer Bildunterschrift oder im Text), versucht Ihr Gehirn, herauszufinden, wie die Wörter und das Bild zusammenhängen, und dadurch feuern mehr Neuronen. Und je mehr Neuronen feuern, umso größer ist die Chance, dass Ihr Gehirn mitbekommt: Bei dieser Sache lohnt es sich, aufzupassen, und vielleicht auch, sich daran zu erinnern.

Ein lockerer Sprachstil hilft, denn Menschen tendieren zu höherer Aufmerksamkeit, wenn ihnen bewusst ist, dass sie ein Gespräch führen – man erwartet dann ja von ihnen, dass sie dem Gespräch folgen und sich beteiligen. Das Erstaunliche daran ist: Es ist Ihrem Gehirn ziemlich egal, dass die »Unterhaltung« zwischen Ihnen und einem Buch stattfindet! Wenn der Schreibstil dagegen formal und trocken ist, hat Ihr Gehirn den gleichen Eindruck wie bei einem Vortrag, bei dem in einem Raum passive Zuhörer sitzen. Nicht nötig, wach zu bleiben.

Aber Abbildungen und ein lockerer Sprachstil sind erst der Anfang.

Das haben WIR getan:

Wir haben ***Bilder*** verwendet, weil Ihr Gehirn auf visuelle Eindrücke eingestellt ist, nicht auf Text. Soweit es Ihr Gehirn betrifft, sagt ein Bild *wirklich* mehr als 1.024 Worte. Und dort, wo Text und Abbildungen zusammenwirken, haben wir den Text *in* die Bilder eingebettet, denn Ihr Gehirn arbeitet besser, wenn der Text *innerhalb* der Sache steht, auf die er sich bezieht, und nicht in einer Bildunterschrift oder irgendwo vergraben im Text.

Wir haben ***Redundanz*** eingesetzt, d.h. dasselbe auf *unterschiedliche* Art und mit verschiedenen Medientypen ausgedrückt, damit Sie es über *mehrere Sinne* aufnehmen. Das erhöht die Chance, dass die Inhalte an mehr als nur einer Stelle in Ihrem Gehirn verankert werden.

Wir haben Konzepte und Bilder in ***unerwarteter*** Weise eingesetzt, weil Ihr Gehirn auf Neuigkeiten programmiert ist. Und wir haben Bilder und Ideen mit zumindest *etwas* ***emotionalem*** *Charakter* verwendet, weil Ihr Gehirn darauf eingestellt ist, auf die Biochemie von Gefühlen zu achten. An alles, was ein *Gefühl* in Ihnen auslöst, können Sie sich mit höherer Wahrscheinlichkeit erinnern, selbst wenn dieses Gefühl nicht mehr ist als ein bisschen ***Belustigung, Überraschung*** oder ***Interesse.***

Wir haben einen ***umgangssprachlichen Stil*** mit direkter Anrede benutzt, denn Ihr Gehirn ist von Natur aus aufmerksamer, wenn es Sie in einer Unterhaltung wähnt, als wenn es davon ausgeht, dass Sie passiv einer Präsentation zuhören – sogar dann, wenn Sie *lesen*.

Wir haben mehr als 80 ***Aktivitäten*** für Sie vorgesehen, denn Ihr Gehirn lernt und behält von Natur aus besser, wenn Sie Dinge ***tun,*** als wenn Sie nur darüber *lesen*. Und wir haben die Übungen zwar anspruchsvoll, aber doch lösbar gemacht, denn so ist es den meisten Lesern am liebsten.

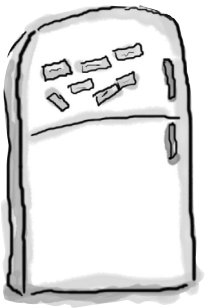
Wir haben ***mehrere unterschiedliche Lernstile*** eingesetzt, denn vielleicht bevorzugen *Sie* ein Schritt-für-Schritt-Vorgehen, während jemand anders erst einmal den groben Zusammenhang verstehen und ein Dritter einfach nur ein Codebeispiel sehen möchte. Aber ganz abgesehen von den jeweiligen Lernvorlieben profitiert *jeder* davon, wenn er die gleichen Inhalte in unterschiedlicher Form präsentiert bekommt.

Wir liefern Inhalte für ***beide Seiten Ihres Gehirns,*** denn je mehr Sie von Ihrem Gehirn einsetzen, umso wahrscheinlicher werden Sie lernen und behalten und umso länger bleiben Sie konzentriert. Wenn Sie mit einer Seite des Gehirns arbeiten, bedeutet das häufig, dass sich die andere Seite des Gehirns ausruhen kann; so können Sie über einen längeren Zeitraum produktiver lernen.

Und wir haben ***Geschichten*** und Übungen aufgenommen, die ***mehr als einen Blickwinkel repräsentieren,*** denn Ihr Gehirn lernt von Natur aus intensiver, wenn es gezwungen ist, selbst zu analysieren und zu beurteilen.

Wir haben ***Herausforderungen*** eingefügt: in Form von Übungen und indem wir ***Fragen*** stellen, auf die es nicht immer eine eindeutige Antwort gibt, denn Ihr Gehirn ist darauf eingestellt, zu lernen und sich zu erinnern, wenn es an etwas *arbeiten* muss. Überlegen Sie: Ihren *Körper* bekommen Sie ja auch nicht in Form, wenn Sie nur die Leute auf dem Sportplatz *beobachten*. Aber wir haben unser Bestes getan, um dafür zu sorgen, dass Sie – wenn Sie schon hart arbeiten – an den *richtigen* Dingen arbeiten. Dass Sie ***nicht einen einzigen Dendriten darauf verschwenden,*** ein schwer verständliches Beispiel zu verarbeiten oder einen schwierigen, mit Fachbegriffen gespickten oder übermäßig gedrängten Text zu analysieren.

Wir haben ***Menschen*** eingesetzt. In Geschichten, Beispielen, Bildern usw. – denn *Sie sind* ein Mensch. Und Ihr Gehirn schenkt *Menschen* mehr Aufmerksamkeit als *Dingen*.



Schneiden Sie dies aus und heften Sie es an Ihren Kühlschrank.

Und das können SIE tun, um sich Ihr Gehirn untertan zu machen

So, wir haben unseren Teil der Arbeit geleistet. Der Rest liegt bei Ihnen. Diese Tipps sind ein Anfang; hören Sie auf Ihr Gehirn und finden Sie heraus, was bei Ihnen funktioniert und was nicht. Probieren Sie neue Wege aus.

1 Immer langsam. Je mehr Sie verstehen, umso weniger müssen Sie auswendig lernen.

Lesen Sie nicht nur. Halten Sie inne und denken Sie nach. Wenn das Buch Sie etwas fragt, springen Sie nicht einfach zur Antwort. Stellen Sie sich vor, dass Sie das wirklich jemand *fragt*. Je gründlicher Sie Ihr Gehirn zum Nachdenken zwingen, umso größer ist die Chance, dass Sie lernen und behalten.

2 Bearbeiten Sie die Übungen. Machen Sie selbst Notizen.

Wir haben sie entworfen, aber wenn wir sie auch für Sie lösen würden, wäre es, als würde jemand anderes Ihr Training für Sie absolvieren. Und sehen Sie sich die Übungen *nicht einfach nur an*. **Benutzen Sie einen Bleistift.** Es deutet vieles darauf hin, dass körperliche Aktivität *beim* Lernen den Lernerfolg erhöhen kann.

3 Lesen Sie die Abschnitte »Es gibt keine dummen Fragen«.

Und zwar alle. Das sind keine Zusatzanmerkungen – ***sie gehören zum Kerninhalt!*** Überspringen Sie sie nicht.

4 Lesen Sie dies als Letztes vor dem Schlafengehen. Oder lesen Sie danach zumindest nichts *Anspruchsvolles* mehr.

Ein Teil des Lernprozesses (vor allem die Übertragung in das Langzeitgedächtnis) findet erst statt, *nachdem* Sie das Buch zur Seite gelegt haben. Ihr Gehirn braucht Zeit für sich, um weitere Verarbeitung zu leisten. Wenn Sie in dieser Zeit etwas Neues aufnehmen, geht ein Teil dessen, was Sie gerade gelernt haben, verloren.

5 Trinken Sie Wasser. Viel.

Ihr Gehirn arbeitet am besten in einem schönen Flüssigkeitsbad. Austrocknung (zu der es schon kommen kann, bevor Sie überhaupt Durst verspüren) beeinträchtigt die kognitive Funktion.

6 Reden Sie drüber. Laut.

Sprechen aktiviert einen anderen Teil des Gehirns. Wenn Sie etwas verstehen oder Ihre Chancen verbessern wollen, sich später daran zu erinnern, sagen Sie es laut. Noch besser: Versuchen Sie, es jemand anderem laut zu erklären. Sie lernen dann schneller und haben vielleicht Ideen, auf die Sie beim bloßen Lesen nie gekommen wären.

7 Hören Sie auf Ihr Gehirn.

Achten Sie darauf, Ihr Gehirn nicht zu überladen. Wenn Sie merken, dass Sie etwas nur noch überfliegen oder dass Sie das gerade erst Gelesene vergessen haben, ist es Zeit für eine Pause. Ab einem bestimmten Punkt lernen Sie nicht mehr schneller, indem Sie mehr hineinzustopfen versuchen; das kann sogar den Lernprozess stören.

8 Aber bitte mit *Gefühl!*

Ihr Gehirn muss wissen, dass es *um etwas Wichtiges geht*. Lassen Sie sich in die Geschichten hineinziehen. Erfinden Sie eigene Bildunterschriften für die Fotos. Über einen schlechten Scherz zu stöhnen, ist *immer noch* besser, als gar nichts zu fühlen.

9 Schreiben Sie viel Code!

Programmieren lernt man nur auf eine Weise: **indem man viel programmiert**! Und das werden Sie in diesem Buch machen. Programmieren ist ein Handwerk. Gut wird man nur durch Übung, und die erhalten Sie hier: Jedes Kapitel enthält eine Menge Übungen, die Sie lösen müssen. Überspringen Sie diese nicht – das Lernen erfolgt zu großen Teilen beim Lösen der Übungen. Zu jeder Übung gibt es eine Lösung – werfen Sie ruhig einen Blick darauf, wenn Sie hängen bleiben! Aber versuchen Sie erst, das Problem zu lösen, bevor Sie sich die Lösung ansehen. Auf alle Fälle sollten Sie die Sache ans Laufen bringen, bevor Sie im Buch weitergehen.

Lies mich

Dies ist ein Lehrbuch, keine Referenz. Wir haben mit Absicht alles weggelassen, was Ihnen dabei in die Quere kommen könnte, das zu lernen, was wir am jeweiligen Punkt des Buchs behandeln. Beim ersten Lesen sollten Sie unbedingt auch am Anfang des Buchs beginnen. Das Buch geht zu jedem Zeitpunkt davon aus, dass Sie bestimmte Dinge bereits gesehen und gelernt haben.

Dies ist nicht Python von Kopf bis Fuß.

In diesem Buch nutzen wir Version 3 der Programmiersprache Python, aber das macht dieses Buch trotzdem nicht zu *Python von Kopf bis Fuß*. Wir haben Python gewählt, weil es gleichermaßen eine ausgezeichnete Programmiersprache für die ersten Schritte in die Programmierung wie für das Erwachsenwerden als Programmierer ist. Es könnte sogar sein, dass Sie nie eine andere Programmiersprache als Python lernen und nutzen müssen (obgleich Ihr Arbeitgeber da eventuell anderer Meinung sein könnte). Aber irgendwo muss man anfangen. Und uns fällt keine geeignetere Programmiersprache als Python ein, wenn es darum geht, das Programmieren zu lernen. Dennoch soll dieses Buch keine Einführung in Python sein. Es ist als Einführung in die *Programmierung* gedacht. Die meisten Dinge, die wir Ihnen hier vorstellen, sollen also die *Programmierkonzepte veranschaulichen*, nicht die Python-Funktionalitäten.

Sie müssen Python 3 auf Ihrem Rechner installieren.

Damit Sie die Programme in diesem Buch ausführen können, müssen Sie Python 3 herunterladen und auf Ihrem Rechner installieren. Das ist nicht so schwer, wie es klingt. Statten Sie der Python-Download-Seite einen Besuch ab und wählen Sie die Option, die für Ihr System am besten geeignet ist. Achten Sie allerdings darauf, dass Sie Version 3 von Python nehmen, *nicht* Version 2: ***http://www.python.org/download***.

Wir beginnen mit den grundlegenden Konzepten der Programmierung, dann lassen wir die Programmierung unmittelbar praktisch werden.

In Kapitel 1 behandeln wir die Grundlagen der Programmierung. Damit können Sie, schon wenn Sie es zu Kapitel 2 geschafft haben, Programme erstellen, die etwas Richtiges, Nützliches und – ja! – Vergnügliches tun. Vermutlich werden Sie erstaunt sein, wie viel man mit den kaum ein Dutzend Codezeilen in Kapitel 2 erreichen kann. Der Rest des Buchs baut auf Ihren kontinuierlich wachsenden Programmierfertigkeiten auf und verwandelt Sie im Handumdrehen vom *Programmieranfänger* zum *Programmierexperten*.

Die Aktivitäten sind NICHT optional.

Die Übungen und Aktivitäten sind kein Beiwerk. Sie sind ein wesentlicher Bestandteil des Buchs. Einige von ihnen sollen Ihnen beim Merken helfen, andere beim Verstehen, und einige helfen Ihnen, das Gelernte anzuwenden. ***Überspringen Sie nichts.***

Die Redundanz ist beabsichtigt und wichtig.

Eins der Dinge, das ein Von Kopf bis Fuß-Buch grundlegend anders macht, ist, dass wir möchten, dass Sie die Sache *wirklich* verstehen. Und wir möchten, dass Sie nach Beendigung des Buchs behalten, was Sie gelernt haben. Einprägen und erinnern sind bei den meisten Lehr- oder Referenzbüchern nicht unbedingt das erste Ziel – in diesem Buch schon. Deswegen werden Ihnen hier einige Konzepte mehrfach begegnen.

Die Codebeispiele sind so schlank wie möglich.

Unsere Leser sagen uns, dass sie es frustrierend finden, sich in 200 Zeilen Code graben zu müssen, um die beiden Zeilen zu finden, die sie wirklich verstehen müssen. Die meisten Beispiele in diesem Buch werden mit so wenig Kontext wie möglich gezeigt, damit der Teil, den Sie lernen sollen, klar und einfach ist. Sie dürfen nicht erwarten, dass der Code robust oder gar vollständig ist. Die Beispiele in diesem Buch wurden speziell für Lehrzwecke geschrieben und sind nicht immer vollständig arbeitsfähig.

Wir haben viele der Codebeispiele in diesem Buch ins Web gestellt, damit Sie die Teile, die Sie benötigen, nicht von Hand eingeben müssen. Die Originale finden Sie an zwei Stellen:

http://www.headfirstlabs.com/books/hfprog/ und

http://programming.itcarlow.ie

Der für die deutsche Ausgabe lokalisierte Code liegt unter

http://examples.oreilly.de/german_examples/hfprogramminger/

Zu den Kopfnuss-Übungen gibt es keine Lösungen.

Bei einigen Übungen gibt es keine richtige Antwort, und bei anderen gehört es zum Lernprozess, den die Kopfnuss-Übungen anstoßen sollen, dass Sie selbst entscheiden müssen, ob und wann Ihre Antworten richtig sind. Manchmal finden Sie auch Hinweise, die Sie in die richtige Richtung lenken.

Die technischen Gutachter

Doug Hellman

Ted Leung

Jeremy Jones

Bill Mietelski

Technische Gutachter:

Doug Hellmann ist Senior Software Engineer bei Racemi und war Cheflektor des *Python Magazine*. Mit Python arbeitet er seit Version 1.4. Zuvor arbeitete er hauptsächlich mit C auf einer Vielzahl von Unix- und Nicht-Unix-Systemen. Die Projekte, an denen er mitgearbeitet hat, reichen von Mapping-Anwendungen bis zur Produktion von Medizinnachrichten, etwas Banking zum Ausgleich eingestreut. In seiner Freizeit arbeitet er an einer Reihe von Open Source-Projekten mit, liest Science-Fiction, Geschichtsbücher und Biografien und schreibt die Blog-Reihe »Python Module of the Week«.

Jeremy Jones ist Koautor von *Python for Unix and Linux System Administration* und arbeitet seit 2001 aktiv mit Python. Er war Entwickler, Systemadministrator, arbeitete in der Qualitätssicherung und im technischen Support. All diese Aufgaben boten Lohn und Herausforderung, doch seine herausforderndste und lohnenswerteste Aufgabe ist sein Dasein als Ehemann und Vater.

Ted Leung programmiert seit 2002 mit Python und ist zurzeit »Principal Software Engineer« bei Sun Microsystems. Er hat einen Bachelor of Science in Mathematik des Massachusetts Institute of Technology und einen Master of Science in Informatik von der Brown University. Teds Weblog finden Sie unter http://www.sauria.com/blog.

Bill Mietelski ist seit 20 Jahren IT-Geek. Aktuell ist er Software-Entwickler bei einer führenden amerikanischen medizinischen Forschungseinrichtung in der Gegend von Chicago und arbeitet an statistischen Studien. Wenn er nicht in seinem Büro sitzt oder anderswo an einen Computer gefesselt ist, finden Sie ihn wahrscheinlich auf dem Golfplatz, wo er kleine weiße Bälle jagt.

Danksagungen

Unser Lektor:

Brian Sawyer war der Lektor für *Programmieren von Kopf bis Fuß*. Wenn er nicht gerade Bücher lektoriert, läuft er Marathon (***nur so zum Vergnügen***). Es stellte sich heraus, dass genau das die richtige Vorbereitung war für den Marathon, als der sich die Produktion dieses Buchs erwiesen hatte. Zeitweise nahm uns Brian ganz ordentlich in die Mangel, was der Qualität der Buchs nur gut getan hat.

Das O'Reilly-Team:

Brett McLaughlin, der Lektor der Reihe, behielt unsere Arbeit stets im Blick und griff mehrfach rettend ein, als wir in Probleme gerieten. **Karen Shaner** gab uns organisatorische Unterstützung und koordinierte mit großem Geschick die technische Begutachtung des Buchs.

Freunde und Kollegen:

Besonders dankbar sind David und Paul **Lou Barr**, die als Erste auf den Gedanken kam, sie beide könnten gemeinsam an diesem Buch arbeiten, die das auch vorschlug und dann hart daran arbeitete, dass diese Idee bei O'Reilly Zustimmung fand. Danke, Lou!

David: Ich danke **Kathy Sierra** und **Bert Bates** für diese außerordentliche Buchreihe. Und dann danke ich noch **Andy Parker** und **Joe Broughton** und **Carl Jacques** und **Simon Jones** und den vielen anderen Freunden, die so wenig von mir hörten, als ich fleißig an diesem Buch schrieb.

Paul: Dank an **Nigel Whyte**, Leiter des Arbeitsbereichs Computing and Networking am Institute of Technology, Carlow, dass er mich bei einem weiteren Schreibprojekt unterstützte. Außerdem hat mein Kollege **Dr. Christophe Meudec** die ersten Entwürfe gelesen und mir sehr willkommene Ermutigung und Verbesserungsvorschläge zukommen lassen. **Joseph Kehoe** hat die ersten Entwürfe ebenfalls in Augenschein genommen und begrüßte, was er sah.

Familie:

David: Mein ganz besonderer Dank gilt meiner Frau, der Autorin von *Statistik von Kopf bis Fuß*, **Dawn Griffiths** – für ihren Verstand, ihren Humor, ihre Geduld und ihre Fähigkeit, vage Ideen in wahre Kapitel zu verwandeln.

Paul: Ich schulde meinem Vater, **Jim Barry**, Dank, der die ersten Entwürfe in Augenschein nahm und (wieder einmal) auf die Stellen hinwies, an denen meine Sprache verbessert und verständlicher gemacht werden könnte. Leider litt mein Privatleben, während dieses Buch meine gesamte Freizeit verschlang. **Deirdre**, **Joseph**, **Aaron** und **Aideen** mussten das Ächzen und Stöhnen, Heulen und Jammern und den einen oder anderen Ausbruch ertragen, wenn der Druck zu groß wurde. Gelegentlich frage ich mich, wie sie es überhaupt mit mir ausgehalten haben, aber irgendwie haben sie es – wofür ich sehr dankbar bin.

Die Unentbehrlichen:

Unsere technischen Gutachter haben ausgezeichnete Arbeit dabei geleistet, uns in der Spur zu halten und sicherzustellen, dass alles, was wir behandelt haben, seine Richtigkeit hat.

Schließlich stehen wir beide tief in der Schuld von Davids Frau **Dawn**, die nicht nur auf David aufpasst, sondern sich auch in die Produktion dieses Buchs einbinden ließ, als es so aussah, als würden wir die Sache nie mehr abschließen. Ohne Dawns Hilfe wäre dieses Buch nie rechtzeitig fertig geworden. Dawn ist der *Programmieren von Kopf bis Fuß*-Schutzengel.

1 Der Anfang des Programmierens

Orientierungshilfe

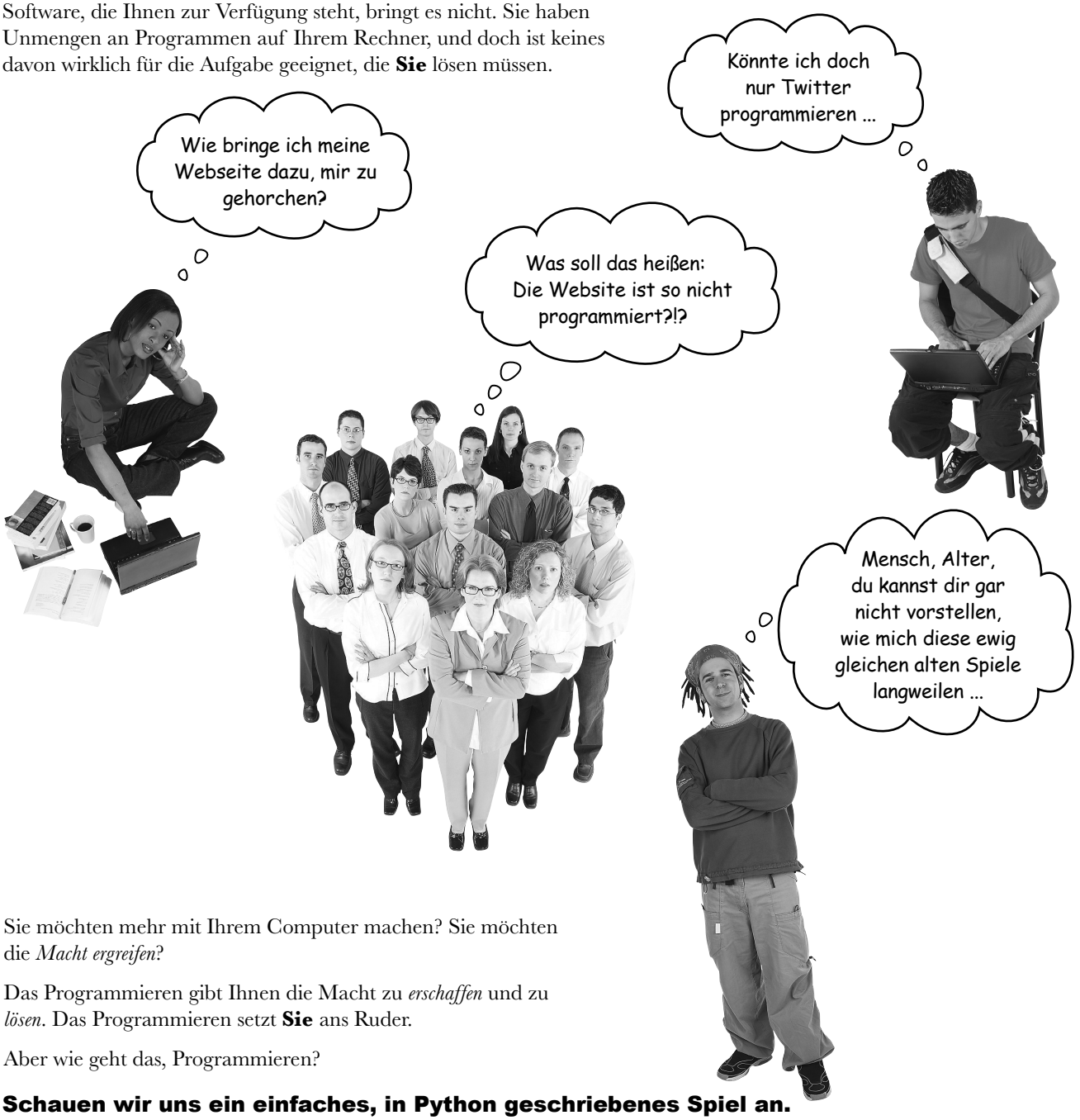
Eigene Programme verleihen Ihnen die Macht, Ihren PC zu steuern.

Fast jeder weiß, wie man mit einem Computer *arbeitet*, aber nur wenige machen den nächsten Schritt und lernen, wie man ihn *steuert*. Wenn Sie Software nutzen, die von anderen entwickelt wurde, sind Sie immer auf das beschränkt, was Sie *deren* Meinung nach tun möchten. Schreiben Sie Ihre eigenen Programme, und nur noch Ihre eigene Vorstellungskraft kann Sie einschränken. Das Programmieren macht Sie kreativer, lässt Sie präziser denken und lehrt Sie, Probleme logisch zu analysieren und zu lösen.

Wollen Sie lieber programmiert oder Programmierer sein?

Mit Programmieren können Sie mehr erreichen

Sie müssen Probleme lösen, um Ihre Arbeit zu erledigen, aber die Software, die Ihnen zur Verfügung steht, bringt es nicht. Sie haben Unmengen an Programmen auf Ihrem Rechner, und doch ist keines davon wirklich für die Aufgabe geeignet, die **Sie** lösen müssen.



Sie möchten mehr mit Ihrem Computer machen? Sie möchten die *Macht ergreifen*?

Das Programmieren gibt Ihnen die Macht zu *erschaffen* und zu *lösen*. Das Programmieren setzt **Sie** ans Ruder.

Aber wie geht das, Programmieren?

Schauen wir uns ein einfaches, in Python geschriebenes Spiel an.

Spitzen Sie Ihren Bleistift

Folgender Code ist ein Programm für ein Ratespiel. Schauen Sie ihn sich sorgfältig an und schreiben Sie neben die Zeilen des Programmtexts, was diese wohl jeweils machen. Es ist überhaupt nicht schlimm, wenn Sie nicht sicher sind, was eine Zeile bewirkt. Versuchen Sie zu raten. Eine Zeile haben wir bereits kommentiert, um Ihnen auf die Sprünge zu helfen:

```
print("Willkommen!")                                    .....

t = input("Erraten Sie die Zahl: ")                     .....

tipp = int(t)                                           Wandelt die Eingabe in eine Zahl um.

if tipp == 5:                                           .....

    print("Gewonnen!")                                  .....

else:                                                   .....

    print("Verloren!")                                  .....

print("Das ist Spiel ist aus!")                         .....
```

Dieser Code wurde in Version 3 der Programmiersprache Python geschrieben. Diese werden wir im gesamten Buch verwenden.

Spitzen Sie Ihren Bleistift
Lösung

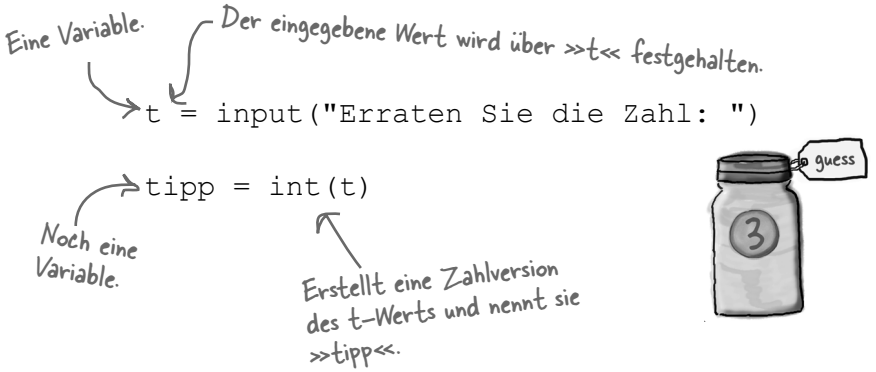
Das ist der Programmtext für ein Ratespiel. Sie sollten aufschreiben, was die Zeilen Ihrer Meinung nach bewirken.

Keine Sorge, wenn Ihre Antworten anders aussehen. Wenn sie unseren ähneln, ist alles in Ordnung.

`print("Willkommen!")`	Eine Begrüßungsmeldung ausgeben.
`t = input("Erraten Sie die Zahl: ")`	Den Benutzer zur Eingabe eines Tipps auffordern.
`tipp = int(t)`	Die Eingabe in eine Zahl umwandeln.
`if tipp == 5:`	Ist der Wert der eingegebenen Zahl gleich 5?
`    print("Gewonnen!")`	>>Gewonnen!<< ausgeben.
`else:`	Andernfalls ...
`    print("Verloren!")`	... >>Verloren!<< ausgeben.
`print("Das Spiel ist aus!")`	Das Spiel beenden.

Aber was sind t und tipp?

Wahrscheinlich fragen Sie sich, was `t` und `tipp` bedeuten. Diese Dinger nennt man **Variablen,** und sie dienen dazu, Daten im Speicher des Computers nachzuhalten.



Eine Variable ist eigentlich nichts anderes als ein **Etikett** für Daten. Gibt der Benutzer auf der Tastatur »3« ein, wird `tipp` auf die Zahl 3 gesetzt. Liest der Computer `tipp`, liest er sie als den Wert 3.

Aufgepasst

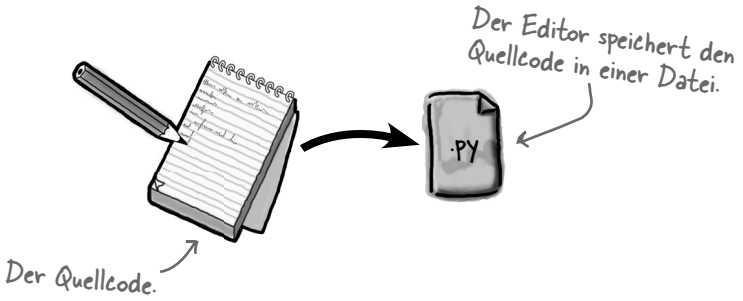
Geben Sie mit den Gleichheitszeichen acht.

Programmiersprachen nutzen =-Zeichen zu unterschiedlichen Zwecken. In den meisten (Python eingeschlossen) stehen zwei aufeinanderfolgende Gleichheitszeichen (==) für einen Test auf Gleichheit. Sie stehen also für: »Sind diese beiden Dinge gleich?« Ein einzelnes Gleichheitszeichen (=) ist hingegen eine Anweisung, die als Zuweisung bezeichnet wird und sagt, dass eine Variable auf ... gesetzt werden soll.

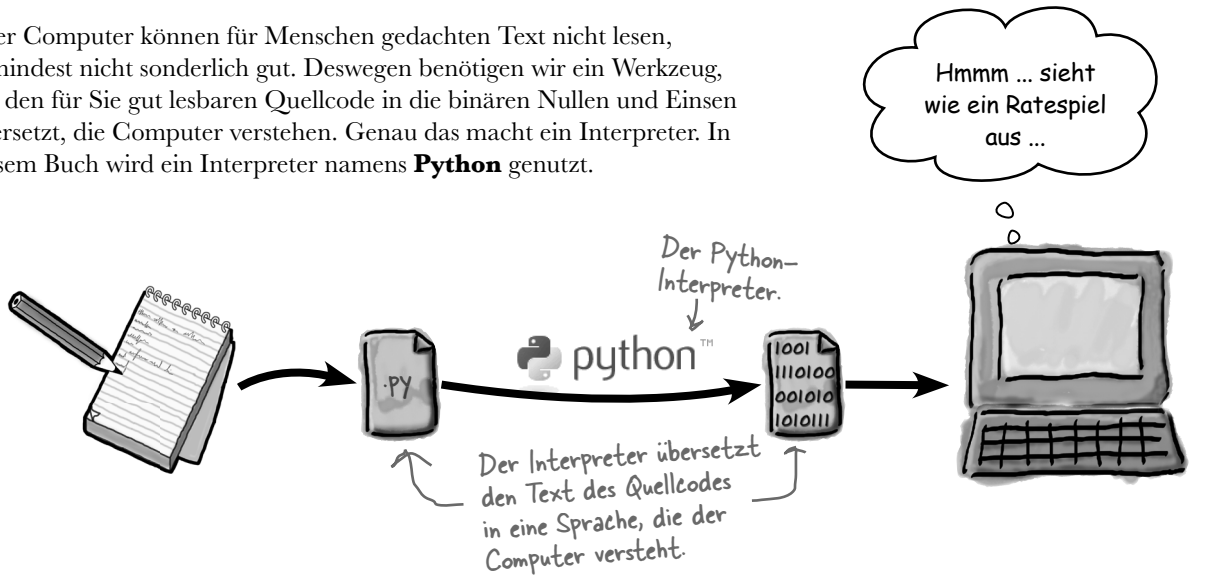
Und wie führt man das Programm aus?

Um das Programm auszuführen, benötigen Sie **zwei** Dinge: einen **Texteditor** und einen **Interpreter**.

Der Editor speichert den von Ihnen geschriebenen Programmtext in einer Datei auf Ihrer Festplatte. Der Programmtext bzw. Quellcode (oder kurz **Code**) ist ganz gewöhnlicher Text, der von Menschen geschrieben und gelesen werden kann.



Aber Computer können für Menschen gedachten Text nicht lesen, zumindest nicht sonderlich gut. Deswegen benötigen wir ein Werkzeug, das den für Sie gut lesbaren Quellcode in die binären Nullen und Einsen übersetzt, die Computer verstehen. Genau das macht ein Interpreter. In diesem Buch wird ein Interpreter namens **Python** genutzt.



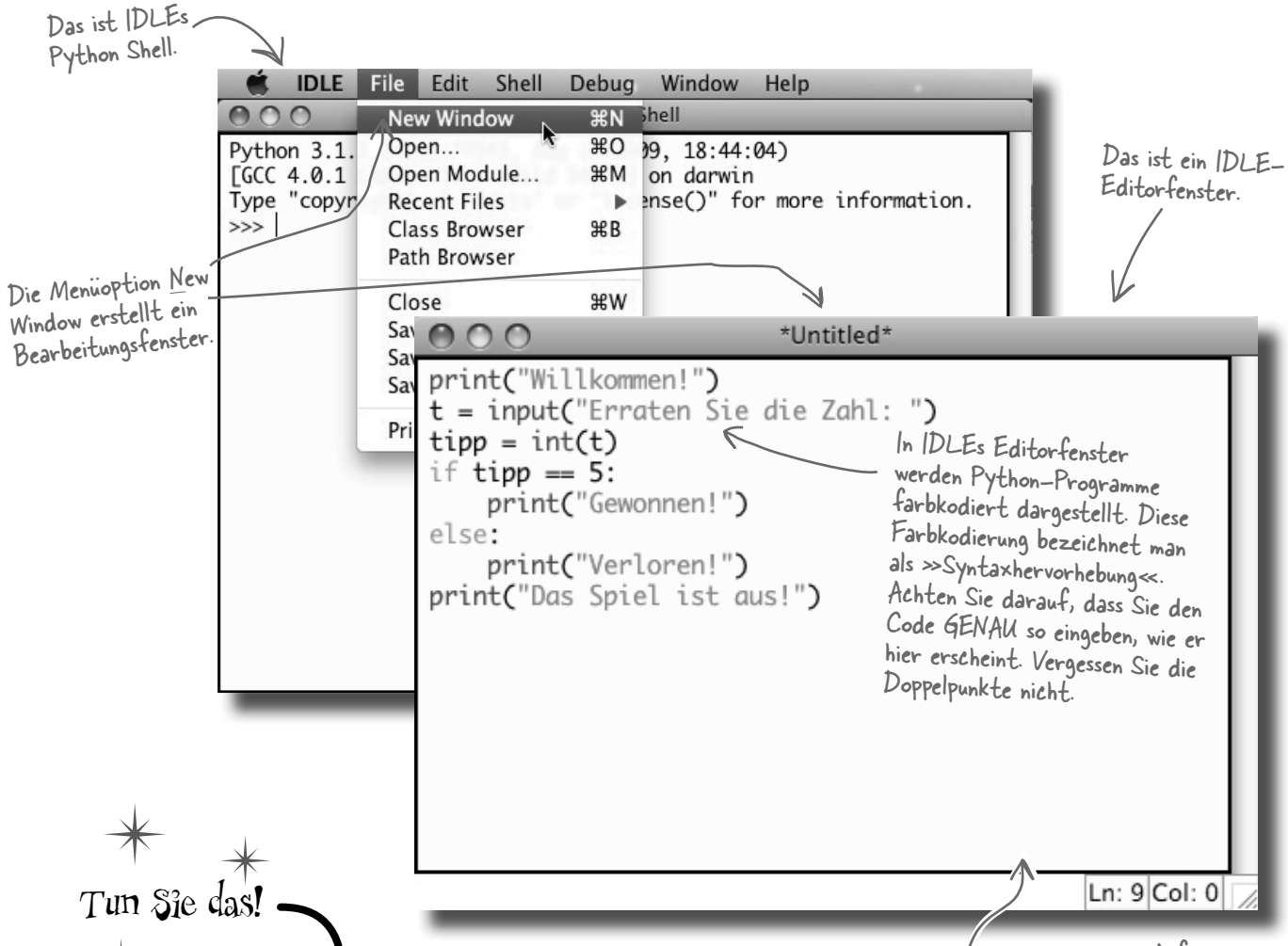
Wir brauchen also einen Editor und einen Python-Interpreter. Glücklicherweise besitzt Python 3 eine eingebaute Anwendung namens **IDLE**, die beides macht und noch etwas mehr. In IDLE können Sie Python-Code schreiben und bearbeiten, außerdem kann es diesen Code in eine binäre Form übersetzen und als Python-Programm ausführen. Deswegen bezeichnet man IDLE als **integrierte Entwicklungsumgebung** (Integrated Development Environment oder kurz IDE).

Schauen wir uns diese Schritte in Aktion an.

Eine neue Programmdatei erstellen

Wenn Sie IDLE starten, wird Ihnen ein Fenster angezeigt, das als **Python Shell** bezeichnet wird. Wählen Sie im Menü `File` die Option `New Window`, um ein neues Bearbeitungsfenster zu öffnen. Geben Sie unseren Programmcode als Text in dieses Fenster in, damit wir loslegen können.

Hinweis: Leider gibt es IDLE nicht in deutscher Lokalisierung.



So sieht IDLE auf unserem Rechner aus. Auf Ihrem könnte es etwas anders erscheinen, aber Menüsystem und Verhalten von IDLE sollten dennoch vollkommen identisch sein, unabhängig davon, welches Betriebssystem Sie nutzen.

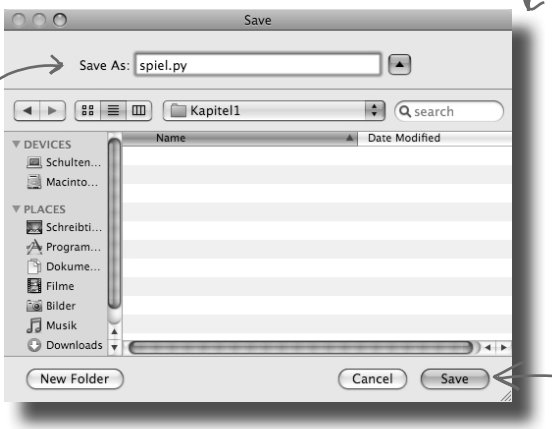
Tun Sie das!

Legen Sie los: Öffnen Sie ein neues IDLE-Editorfenster und geben Sie den Code von Seite 3 ein.

Code vorbereiten und ausführen

Der nächste Schritt ist, Ihren Programmcode für die Ausführung *vorzubereiten*. Wählen Sie dazu `File` → `Save` im Menü, um den Code in einer Datei zu speichern. Vergeben Sie einen geeigneten Namen für Ihr Programm.

Wenn Sie File → Save wählen, können Sie Ihren Code in einer Datei speichern.



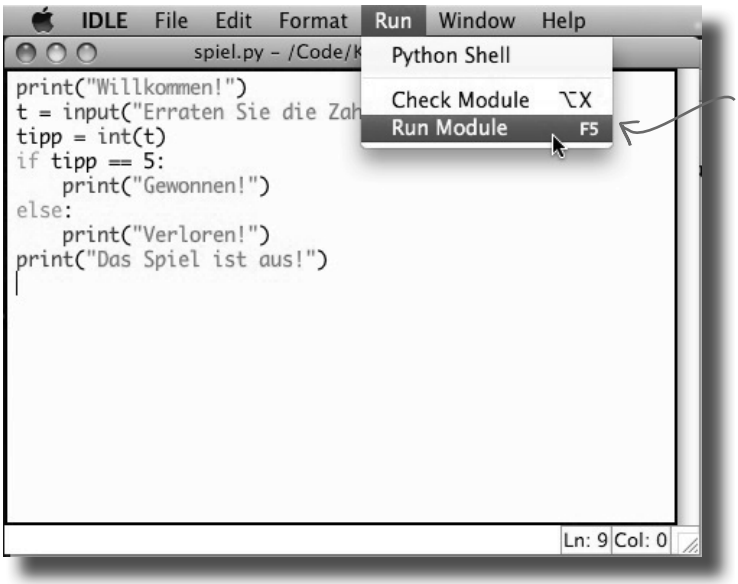
Die Namen von Python-Programmdateien enden üblicherweise mit >>.py<<.

Klicken Sie auf den Button >>Save<<, um die Datei zu erstellen und zu speichern.

Python-Programme werden in der Regel in Dateien mit der Dateinamenserweiterung `.py` gespeichert. Nennen Sie das Programm also `spiel.py`.

IDLE ist es vollkommen gleich, in welchem Verzeichnis Sie die Datei speichern. Viele Programmierer ziehen es vor, für jedes Programmierprojekt ein eigenes Verzeichnis zu erstellen. Für den Augenblick reicht es aber, wenn Sie den Code in einem Verzeichnis speichern, das Sie sich leicht merken können.

Schauen wir jetzt, was passiert, wenn wir das Programm ausführen.



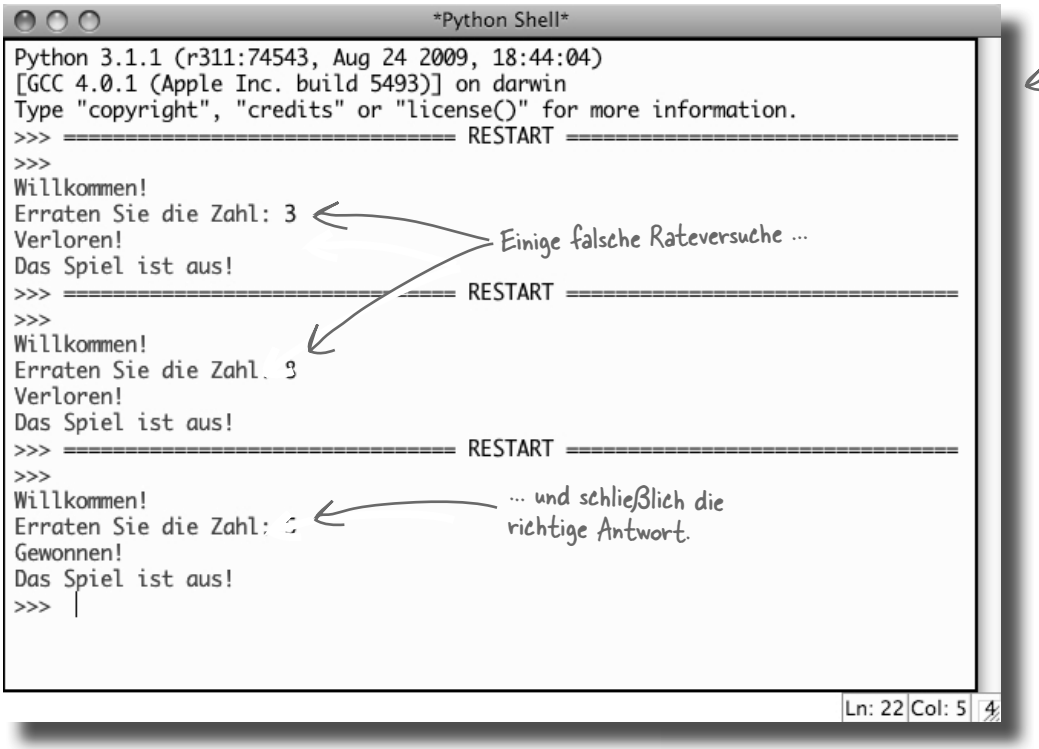
Gehen Sie im Bearbeitungsfenster von IDLE zum Menü >>Run<< und wählen Sie >>Run Module<<.

PROBEFAHRT

Wenn Sie das Programm ausführen wollen, muss das Editorfenster mit dem Programmcode für `spiel.py` aktiv sein. Immer wenn Sie ein Programm ausführen (oder erneut ausführen) wollen, müssen Sie dazu auf das IDLE-Bearbeitungsfenster klicken und darin die Option `Run Module` aus dem Menü `Run` wählen. Das Wort ***Module*** ist ein Name, den IDLE nutzt, um auf Ihren Programmcode zu verweisen.

Wenn Sie Ihren Code in IDLE ausführen, erscheinen alle Meldungen in der Python Shell, nicht im Bearbeitungsfenster. IDLE macht die Shell automatisch zum aktuellen Fenster, wenn Sie ein Programm ausführen lassen.

Folgendes passiert, wenn Sie den Code ausführen:



Glückwunsch! Das Programm funktioniert.

Jedes Mal, wenn Sie den Code ausführen, zeigt das Programm die Nachricht »Willkommen!« an, wartet auf eine Benutzereingabe über die Tastatur und sagt dann, ob der richtige Wert erraten wurde. Das bedeutet, dass das Programm eine **Eingabe** entgegennimmt, Daten **verarbeitet** und dann eine **Ausgabe** generiert.

Es gibt keine Dummen Fragen

F: Von Python habe ich noch nie gehört. Ist es beliebt?

A: Python wird in vielen coolen Unternehmen genutzt. Beispielsweise nutzen Google, Industrial Light & Magic, YouTube und die NASA (um nur einige zu nennen) Python. Wir nehmen mal an, dass man dort weiß, was man tut.

F: Und wenn ich mit diesem Buch durch bin, kann ich Python über Bord werfen und etwas anderes lernen, C# oder Java beispielsweise?

A: Nur wenn Sie wollen. Es kann gut sein, dass Sie nie eine andere Programmiersprache als Python benötigen werden. Aber wenn Sie unbedingt eine weitere Programmiersprache lernen wollen, können Sie alles, was Sie in diesem Buch über das Programmieren lernen, mit wenig Aufwand in anderen Sprachen zur Anwendung bringen.

F: Aber einer meiner Kumpel hat mir gesagt, ich sollte besser Java oder C# lernen. Warum nutzen Sie keins davon in diesem Buch?

A: Java und C# sind großartige Programmiertechnologien, können aber schwer zu erlernen sein, insbesondere wenn man gerade erst mit dem Programmieren beginnt. Das ist bei Python nicht der Fall. Dieses Buch soll Ihnen beibringen, wie man programmiert, und wenn wir Python als erste Programmiersprache nutzen, gelingt uns das umso leichter.

F: Offensichtlich gibt es viele verschiedene Versionen von Python. Welche sollte ich nutzen?

A: Es gibt zwei relevante Versionen von Python: 2 und 3. Dieses Buch basiert auf Version 3 der Sprache. Python 3 ist die Zukunft der Sprache; neue Funktionen werden Python 3 mit Sicherheit hinzugefügt, Python 2 wahrscheinlich nicht. Natürlich ist Python 3 wie alle anderen Versionen kostenlos. Man muss sich also nicht überlegen, ob man sich leisten kann, damit zu arbeiten.

F: Läuft Python auch auf meinem Handy – wie Java?

A: Das hängt von Ihrem Handy ab. Python wurde so entworfen, dass es unter vielen verschiedenen Technologien und Betriebssystemen läuft. Eigenen Code auf dem eigenen Handy auszuführen, ist eine sehr spezielle Anforderung, die Java zurzeit ausgezeichnet abdeckt. Als Programmiertechnologie wurde Java ursprünglich zur Verwendung auf Kleingeräten entworfen, es ist also kein Wunder, dass es eine gute, geeignete und beliebte Option ist, wenn es um die Programmierung von Handys geht.

F: Warum heißt die Python-IDE IDLE?

A: In gewisser Hinsicht wohl, weil es ähnlich klingt wie IDE. Aber wir vermuten, dass es mehr mit Eric Idle zu tun hat, einem der Gründungsmitglieder der Comedy-Gruppe Monty Python's Flying Circus.

F: Wie bitte?!? Monty Python's Flying... Was?

A: Circus. Ja, wissen wir: Klingt etwas verrrückt, oder? Und glauben Sie uns, das ist es. Aber auch lustig. Der Schöpfer von Python, Guido van Rossum, ist ein großer Monthy-Python-Fan, und man sagt, er habe Wiederholungen der Sendungen gesehen, während er Python entwarf. Sie werden in der Python-Gemeinschaft auf eine Menge Monty-Python-Anspielungen stoßen. Tote Papageien sind besonders beliebt.

F: Was bedeutet das int(t)?

A: Es sagt Python, dass die Eingabe des Benutzers als Zahl, nicht als Buchstabe interpretiert werden soll. Bei Programmiersprachen ist die Zahl 5 etwas anderes als der Buchstabe ›5‹.

F: Und was ist, wenn wir das weglassen würden?

A: Dann hätte der Computer die Eingabe des Benutzers als Buchstabe behandelt. Wenn Sie den Computer fragen, ob eine Zahl gleich einem Buchstaben ist, reagiert er irritiert und sagt Ihnen, dass sie das nicht ist.

F: Warum das?

A: Weil der Computer, wenn er meint, dass zwei Daten unterschiedlichen »Typs« sind, davon ausgeht, dass es keine Möglichkeit gibt, sie zu vergleichen.

F: Und was wäre gewesen, wenn ich *keine* Zahl eingegeben hätte, als ich aufgefordert wurde, einen Tipp einzugeben? Was ist, wenn ich einfach meinen Namen oder etwas anderes eingegeben hätte?

A: Die Ausführung des Codes wäre mit einer Fehlermeldung abgebrochen worden. Python würde Ihnen sagen, dass das Programm mit einem »ValueError« abgebrochen wurde. (Über diese und andere Fehlermeldungen werden Sie später in diesem Buch noch mehr erfahren.).

Einer Ihrer Nutzer.

Ich raff das nicht. Wie, bitte, soll ich denn auf die zu erratende Zahl kommen? Das blöde Programm sagt mir doch nur, ob mein Tipp falsch oder richtig ist. Kann es uns denn nicht irgendeinen Hinweis geben?

Das Programm muss mehr leisten.

Bislang sagt das Ratespiel dem Benutzer nur, ob sein Tipp richtig oder falsch ist. Mehr nicht. Es wäre hilfreicher, wenn das Programm aussagekräftigere Meldungen anzeigen würde, beispielsweise ob der Tipp **höher** oder **niedriger** ist als die richtige Antwort. Das würde dem Benutzer helfen, der richtigen Antwort näher zu kommen, wenn das Programm das nächste Mal ausgeführt wird.

Erreichen können wir das, indem wir den Code ändern. Aber auf welche Weise?

Dieses Programm muss Meldungen anzeigen, die informativer sind.

spiel.py – /Code/Kapitel1/spiel.py

```
print("Willkommen!")
t = input("Erraten Sie die Zahl: ")
tipp = int(t)
if tipp == 5:
    print("Gewonnen!")
else:
    print("Verloren!")
print("Das Spiel ist aus!")
```

Ln: 9 Col: 0

Spitzen Sie Ihren Bleistift

Sie müssen entscheiden, welche Meldungen dem Benutzer angezeigt werden sollen. Unten sehen Sie eine Tabelle mit einigen Werten, die Benutzer eingeben könnten. Was müsste die Meldung jeweils sagen?

Programmänderung

Eingegebene Zahl	Anzuzeigende Meldung
3	
5	
7	
8	

KOPF-NUSS

Schauen Sie sich den ursprünglichen Code an. Sie brauchen mehr als bloß ein paar `print()`-Anweisungen, um informativere Meldungen auszugeben. Was sonst werden Sie noch benötigen?

Spitzen Sie Ihren Bleistift
Lösung

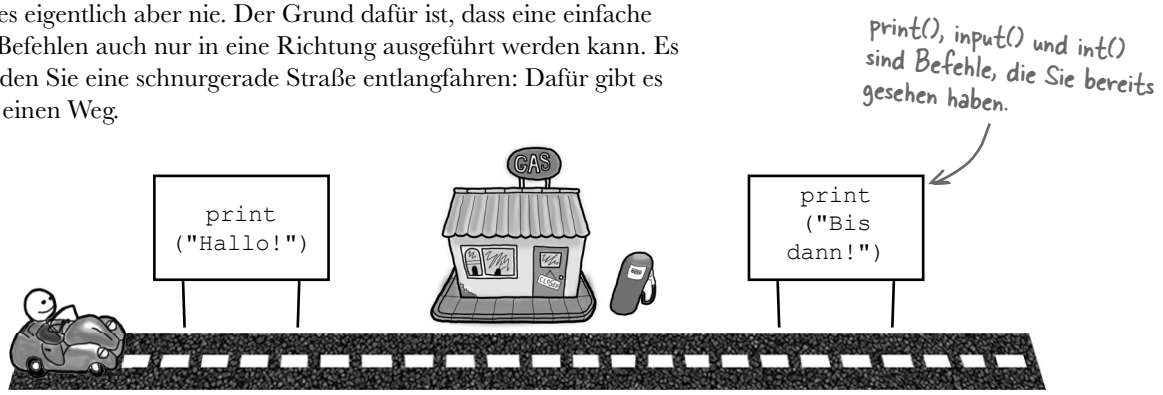
Sie müssen entscheiden, welche Meldungen dem Benutzer angezeigt werden sollen. Unten sehen Sie eine Tabelle mit einigen Werten, die Benutzer eingeben könnten. Was müsste die Meldung jeweils sagen?

Programmänderung

Eingegebene Zahl	Anzuzeigende Meldung
3	Zu klein
5	Gewonnen!
7	Zu groß
8	Zu groß

Ein Programm ist mehr als eine einfache Folge von Befehlen

Man *könnte* ein Programm erstellen, das eine **einfache** Folge von Befehlen ist, macht es eigentlich aber nie. Der Grund dafür ist, dass eine einfache Folge von Befehlen auch nur in eine Richtung ausgeführt werden kann. Es ist, als würden Sie eine schnurgerade Straße entlangfahren: Dafür gibt es genau nur einen Weg.



Aber Programme müssen viel schlauer sein.