O'REILLY®

Leicht verständlich
Mit vielen Illustrationen und Beispielen

Mathe-Basics für Data Scientists

Lineare Algebra, Statistik und Wahrscheinlichkeitsrechnung für die Datenanalyse

Thomas Nield

Übersetzung von Frank Langenau

Lob für »Mathe-Basics für Data Scientists«

In der Kakofonie, die die derzeitige Bildungslandschaft im Bereich der Data Science darstellt, hebt sich dieses Buch als Ressource mit vielen klaren, praktischen Beispielen für die Grundlagen ab, die erforderlich sind, um Daten zu verstehen und mit ihnen zu arbeiten. Da es in diesem Buch um die Basics geht, wird der Leser in die Lage versetzt, alle Aufgaben im Bereich Data Science mit einem stabilen Hintergrundwissen zu ihren Bausteinen zu bewältigen.

– Vicki Boykis, Senior Machine Learning Engineer bei Tumblr

Data Science baut auf linearer Algebra, Wahrscheinlichkeitstheorie und Analysis auf. Thomas Nield führt uns fachkundig durch all diese Themen – und mehr –, um eine solide Grundlage für das Verständnis der mathematischen Instrumente der Data Science zu schaffen.

– Mike X Cohen, sincXpress

Als Data Scientist verwenden wir täglich anspruchsvolle Modelle und Algorithmen. Dieses Buch entmystifiziert die Mathematik, die dahintersteckt, sodass sich diese Modelle und Algorithmen leichter begreifen und implementieren lassen.

– Siddharth Yadav, freiberuflicher Data Scientist

Ich wünschte, ich hätte früher schon mit diesem Buch arbeiten können! Thomas Nield hat es geschafft, komplexe mathematische Themen auf eine leicht verdauliche und fesselnde Art zu erklären. Ein erfrischender Ansatz sowohl für die Mathematik als auch die Data Science – er erklärt nahtlos die grundlegenden mathematischen Konzepte und ihre unmittelbaren Anwendungen beim maschinellen Lernen. Dieses Buch ist Pflichtlektüre für alle angehenden Data Scientists.

– Tatiana Ediger, freiberuflich tätig als Data Scientist, Kursentwicklerin und Dozentin

Mathe-Basics für Data Scientists

Mathe-Basics für Data Scientists

Lineare Algebra, Statistik
und Wahrscheinlichkeitsrechnung
für die Datenanalyse

Thomas Nield

Deutsche Übersetzung von
Frank Langenau

O'REILLY®

Thomas Nield

Lektorat: Alexandra Follenius
Übersetzung: Frank Langenau
Copy-Editing: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner, Frank Heidt
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*
Druck und Bindung: mediaprint solutions 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 978-3-96009-215-5
PDF 978-3-96010-764-4
ePub 978-3-96010-765-1
mobi 978-3-96010-766-8

1. Auflage 2024

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Essential Math for Data Science*, ISBN 9781098102937 © 2022 Thomas Nield. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde mit mineralölfreien Farben auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie. Hergestellt in Deutschland.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *kommentar@oreilly.de*.

5 4 3 2 1 0

Inhalt

Einführung

In den letzten zehn Jahren hat das Interesse an angewandter Mathematik und Statistik in unserem Arbeits- und Lebensalltag zugenommen. Warum ist das so? Hat es mit dem gestiegenen Interesse an *Data Science* zu tun, das von der Harvard Business Review als der »sexiest Job des 21. Jahrhunderts« deklariert wurde (*https://oreil.ly/GslO6*)? Oder ist es das Versprechen, dass maschinelles Lernen und *künstliche Intelligenz* unser Leben verändern werden? Liegt es daran, dass die Schlagzeilen mit Studien, Umfragen und Forschungsergebnissen überschwemmt werden, wir aber nicht wissen, wie wir solche Behauptungen überprüfen sollen? Oder ist es das Versprechen, dass »selbstfahrende« Autos und Roboter in naher Zukunft Arbeitsplätze automatisieren werden?

Ich behaupte, dass die Disziplinen Mathematik und Statistik aufgrund der zunehmenden Verfügbarkeit von Daten das breite Interesse der Öffentlichkeit geweckt haben und wir Mathematik, Statistik und maschinelles Lernen benötigen, um diese Daten sinnvoll zu nutzen. Ja, wir haben wissenschaftliche Werkzeuge, maschinelles Lernen und andere Instrumente der Automatisierung, die wie die Sirenen nach uns rufen. Diesen »Blackboxes«, Geräten und Programmen vertrauen wir blindlings; wir verstehen sie zwar nicht, aber wir nutzen sie trotzdem.

Man ist zwar geneigt zu glauben, dass Computer schlauer sind als wir (und dieser Gedanke wird häufig vermarktet), aber die Realität ist weit davon entfernt. Diese Diskrepanz kann auf vielen Ebenen heikel sein. Wollen Sie wirklich, dass ein Algorithmus oder eine KI eine Strafe verhängt oder ein Fahrzeug steuert, dass aber niemand, nicht einmal der Entwickler, erklären kann, wie es zu einer bestimmten Entscheidung gekommen ist? Erklärbarkeit ist die nächste Stufe der statistischen Datenverarbeitung und der KI. Diese kann erst dann beginnen, wenn wir die Blackbox öffnen und die Mathematik enthüllen.

Vielleicht fragen Sie sich auch, wieso eine Entwicklerin nicht wissen kann, wie ihr eigener Algorithmus funktioniert. Darauf gehen wir in der zweiten Hälfte des Buchs ein, wenn wir Techniken des maschinellen Lernens erörtern und aufzeigen, warum wir die Mathematik hinter den von uns erstellten Blackboxes verstehen müssen.

Ein weiterer Punkt ist, dass der Grund für die massenhafte Datenerfassung vor allem in den vernetzten Geräten und ihrer Präsenz in unserem Alltag liegt. Wir nutzen das Internet nicht mehr nur über einen Desktop- oder Laptop-Computer. Wir haben es jetzt auch in unseren Smartphones, Autos und Haushaltsgeräten dabei. Dies hat in den letzten zwei Jahrzehnten einen fast unmerklichen Wandel in Gang gesetzt. Daten haben sich von einem operativen Werkzeug zu etwas entwickelt, das für Ziele gesammelt und analysiert wird, die noch gar nicht genau umrissen sind. Eine Smartwatch sammelt ständig Daten über unsere Herzfrequenz, die Atmung, die zu Fuß zurückgelegte Strecke und andere Kennwerte. Dann lädt sie die Daten hoch in eine Cloud, um sie zusammen mit anderen Usern zu analysieren. Unsere Fahrgewohnheiten werden von computergesteuerten Autos erfasst und von den Herstellern genutzt, um Daten für selbstfahrende Fahrzeuge zusammenzutragen. Sogar »intelligente Zahnbürsten« finden ihren Weg in die Drogerien. Sie zeichnen die Putzgewohnheiten auf und speichern diese Daten in der Cloud. Es lässt sich darüber streiten, ob die Daten der intelligenten Zahnbürste nützlich und wichtig sind!

Dieses ganze Datensammeln durchdringt jeden Winkel unseres Lebens. Das kann erdrückend sein, und man könnte ein ganzes Buch über Bedenken hinsichtlich Datenschutz und Ethik schreiben. Aber die Verfügbarkeit von Daten bietet auch die Möglichkeit, Mathematik und Statistik auf neue Art und Weise zu nutzen und sich außerhalb des akademischen Umfelds stärker zu engagieren. Wir können mehr über die menschliche Erfahrung lernen, das Design und die Anwendung von Produkten verbessern sowie kommerzielle Strategien optimieren. Wenn Sie die in diesem Buch vorgestellten Ideen verstehen, werden Sie den Wert erschließen können, der in unserer Daten sammelnden Infrastruktur steckt. Das bedeutet nicht, dass Daten und statistische Werkzeuge ein Allheilmittel sind, um alle Probleme dieser Welt zu lösen, aber sie bieten uns neue Werkzeuge, die wir nutzen können. Manchmal ist es ebenso wertvoll, bestimmte Datenprojekte als Irrwege anzuerkennen und sich klarzumachen, dass die Anstrengungen an anderer Stelle besser investiert sind.

Die zunehmende Verfügbarkeit von Daten hat bewirkt, dass Data Science und maschinelles Lernen zu gefragten Berufen geführt haben. Wir definieren die Mathe-Basics als einen Umgang mit Wahrscheinlichkeit, linearer Algebra, Statistik und maschinellem Lernen. Wenn Sie eine Karriere in den Bereichen Data Science, maschinelles Lernen oder Softwaretechnik anstreben, sind diese Themen unerlässlich. Ich werde nur so viel Hochschulmathematik, Analysis und Statistik einbringen, wie es für das Verständnis der Blackbox-Bibliotheken – mit denen Sie arbeiten werden – unerlässlich ist.

Mit diesem Buch möchte ich Leserinnen und Lesern verschiedene Bereiche der Mathematik, Statistik und des maschinellen Lernens näherbringen, die auf reale Probleme anwendbar sind. Die ersten vier Kapitel befassen sich mit grundlegenden mathematischen Konzepten, darunter praktische Analysis, Wahrscheinlichkeit, lineare Algebra und Statistik. Die letzten drei Kapitel sind dem maschinellen Lernen gewid-

met. Der ultimative Zweck eines Lehrbuchs über maschinelles Lernen ist es, alles zu integrieren, was wir lernen, und praktische Einblicke in die Verwendung von maschinellem Lernen und statistischen Bibliotheken über ein Blackbox-Verständnis hinaus zu demonstrieren.

Um den Beispielen zu folgen, benötigen Sie lediglich einen Windows-/Mac-/Linux-Computer und eine Python-3-Umgebung Ihrer Wahl. Die wichtigsten Python-Bibliotheken, die Sie benötigen werden, sind `numpy`, `scipy`, `sympy` und `sklearn`. Wenn Sie mit Python nicht vertraut sind: Keine Angst, es handelt sich um eine einfache und übersichtliche Programmiersprache, die sich der Unterstützung durch riesige Lernressourcen erfreut. Unter anderem empfehle ich folgende Quellen:

Einführung in Data Science: Grundprinzipien der Datenanalyse mit Python, 2. Auflage, von Joel Grus (O'Reilly)
: Das zweite Kapitel dieses Buchs ist der beste Crashkurs in Python, der mir je untergekommen ist. Selbst wenn Sie noch nie zuvor ein Programm geschrieben haben, gelingt es Joel auf fantastische Art und Weise, Ihnen in kürzester Zeit einen effektiven Einstieg in Python zu ermöglichen. Zudem ist es ein großartiges Buch, das man im Regal stehen hat und mit dem man sein mathematisches Wissen anwenden kann!

Python for the Busy Java Developer von Deepak Sarda (Apress)
: Wenn Sie als Softwareentwicklerin oder Softwareentwickler von einem statisch typisierten, objektorientierten Programmierhintergrund kommen, sollten Sie zu diesem Buch greifen. Als jemand, der mit Java zu programmieren begonnen hat, schätze ich es sehr, wie Deepak die Features von Python insbesondere für Java-Entwickler erörtert. Sollten Sie mit .NET, C++ oder anderen C-ähnlichen Sprachen gearbeitet haben, werden Sie Python mit diesem Buch aller Voraussicht nach effektiv lernen.

Dieses Buch wird Sie nicht zu einem Experten machen oder Ihnen einen Doktortitel verleihen. Nach Möglichkeit vermeide ich mathematische Ausdrücke, die mit griechischen Buchstaben gespickt sind, und werde sie stattdessen in einfacher Sprache beschreiben. Aber dieses Buch wird Ihnen den Umgang mit Mathematik und Statistik erleichtern und Ihnen das *wesentliche* Wissen vermitteln, damit Sie sich in diesen Bereichen erfolgreich bewegen können. Meiner Ansicht nach besteht der Weg zum Erfolg nicht darin, ein profundes Spezialwissen zu einem Thema zu haben, sondern sich mit mehreren Themen auseinanderzusetzen und praktische Kenntnisse zu erwerben. Das ist das Ziel dieses Buchs, und Sie werden gerade genug lernen, um angriffslustig zu sein und diese einst so schwer fassbaren kritischen Fragen zu stellen.

Fangen wir also an!

Konventionen in diesem Buch

In diesem Buch werden folgende typografische Konventionen verwendet:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Schreibmaschinenschrift`
: Wird in Programmlistings verwendet und im Fließtext für Programmelemente wie zum Beispiel Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Schreibmaschinenschrift fett`
: Kennzeichnet Befehle oder andere Texte, die vom Benutzer buchstäblich eingegeben werden sollen.

`Schreibmaschinenschrift kursiv`
: Zeigt Text, der ersetzt werden soll durch Werte, die der Benutzer bereitstellt, oder Werte, die sich aus dem Kontext ergeben.

Tipp
Dieses Element kennzeichnet einen Tipp oder Vorschlag.

Hinweis
Dieses Element kennzeichnet einen allgemeinen Hinweis.

Warnung
Dieses Element steht für eine Warnung oder erhöhte Aufmerksamkeit.

Codebeispiele

Ergänzendes Material (Codebeispiele, Übungen usw.) stehen Ihnen unter *https://github.com/thomasnield/machine-learning-demo-data* zum Download zur Verfügung.

Dieses Buch soll Ihnen bei Ihrer täglichen Arbeit helfen. Falls Beispielcode zum Buch angeboten wird, dürfen Sie ihn im Allgemeinen in Ihren Programmen und für Dokumentationen verwenden. Sie müssen uns nicht um Erlaubnis bitten, es sei denn, Sie kopieren einen erheblichen Teil des Codes. Wenn Sie zum Beispiel ein Programm schreiben, das einige Codeblöcke aus diesem Buch verwendet, benötigen Sie keine Erlaubnis. Sollten Sie aber Beispiele aus O'Reilly-Büchern verkaufen

oder verbreiten, ist eine Erlaubnis erforderlich. Wenn Sie eine Frage beantworten und dabei dieses Buch oder Beispielcode aus diesem Buch zitieren, brauchen Sie wiederum keine Erlaubnis. Aber lassen Sie erhebliche Teile des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts einfließen, ist eine Erlaubnis einzuholen.

Wir schätzen eine Quellenangabe, verlangen sie aber nicht. Eine Quellenangabe umfasst in der Regel Titel, Autor, Verlag und ISBN, zum Beispiel: »*Mathe-Basics für Data Scientists* von Thomas Nield, O'Reilly 2024, ISBN 978-3-96009-215-5.«

Wenn Sie der Meinung sind, dass Sie die Codebeispiele in einer Weise verwenden, die über die oben erteilte Erlaubnis hinausgeht, kontaktieren Sie uns bitte unter *kommentar@oreilly.de*.

Danksagungen

An diesem Buch haben über ein Jahr lang viele Menschen mitgearbeitet. Zunächst möchte ich meiner Frau Kimberly für ihre Unterstützung danken, während ich dieses Buch geschrieben habe, insbesondere in der Zeit bis zum ersten Geburtstag unseres Sohnes Wyatt. Kimberly ist eine großartige Ehefrau und Mutter, und alles, was ich jetzt tue, ist für meinen Sohn und die bessere Zukunft unserer Familie.

Ich möchte meinen Eltern dafür danken, dass sie mich gelehrt haben, über meine Grenzen hinaus zu kämpfen und niemals das Handtuch zu werfen. Dem Thema dieses Buchs entsprechend bin ich froh, dass sie mich ermutigt haben, mich auf der Highschool und am College ernsthaft mit Analysis zu beschäftigen, und niemand kann ein Buch schreiben, ohne regelmäßig seine Komfortzone zu verlassen.

Dem fantastischen Team von Lektorinnen und Lektoren und Mitarbeitern bei O'Reilly möchte ich danken, die mir immer wieder Türen geöffnet haben, seit ich 2015 mein erstes Buch über SQL geschrieben habe. Es war fantastisch, mit Jill Leonard und Jessica Haberman zusammenzuarbeiten, um dieses Buch zu schreiben und zu veröffentlichen. Zudem bin ich dankbar, dass Jess an mich gedacht hat, als dieses Thema aufkam.

Ein Dank geht an meine Kollegen an der University of Southern California (USC) im Programm für Flugsicherheit. Die Möglichkeit, Pionierarbeit im Bereich der Sicherheit von Systemen mit künstlicher Intelligenz zu leisten, hat mir Einsichten vermittelt, die nur wenige Menschen haben, und ich freue mich darauf, zu sehen, was wir in den kommenden Jahren noch erreichen werden. Arch, Sie versetzen mich immer wieder in Erstaunen, und ich befürchte, dass die Welt an dem Tag, an dem Sie in den Ruhestand gehen, aufhören wird zu funktionieren.

Zu guter Letzt möchte ich meinem Bruder Dwight Nield und meinem Freund Jon Ostrower danken, die Partner in meinem Unternehmen Yawman Flight sind. Ein Start-up zu gründen, ist schwer, und ihre Hilfe hat mir wertvolle Zeit verschafft, um dieses Buch zu schreiben. Jon hat mich an der USC an Bord geholt, und seine unermüdlichen Leistungen in der Welt des Luftfahrtjournalismus sind schlichtweg

bemerkenswert (suchen Sie nach seinem Namen!). Es ist mir eine Ehre, dass sie eine Erfindung, die ich in meiner Garage begonnen habe, genauso leidenschaftlich verfolgen wie ich, und ich glaube nicht, dass ich diese ohne sie in die Welt bringen könnte.

Allen, die ich hier nicht erwähnt habe, danke ich für die großen und kleinen Dinge, die ihr getan habt. Meistens bin ich dafür belohnt worden, dass ich neugierig war und Fragen gestellt habe. Das halte ich nicht für selbstverständlich. Wie Ted Lasso sagte: »Sei neugierig, nicht voreingenommen.«

KAPITEL 1

Grundlegende Mathematik und Infinitesimalrechnung

Das erste Kapitel beginnen wir mit der Frage, was Zahlen sind und wie Variablen und Funktionen in einem kartesischen System funktionieren. Dann beschäftigen wir uns mit Potenzen und Logarithmen. Anschließend geht es um die beiden grundlegenden Operationen der Infinitesimalrechnung: Ableitungen und Integrale.

Bevor wir in die angewandten Bereiche der grundlegenden Mathematik – zum Beispiel Wahrscheinlichkeit, lineare Algebra, Statistik und maschinelles Lernen – eintauchen, ist es zweckmäßig, einige grundlegende Konzepte der Mathematik und Infinitesimalrechnung aufzufrischen. Doch das sollte nicht dazu führen, dass Sie dieses Buch fallen lassen und schreiend davonlaufen. Keine Sorge! Ich werde Ihnen zeigen, wie man Ableitungen und Integrale für eine Funktion auf eine Art und Weise berechnet, die Ihnen im Studium wahrscheinlich nicht beigebracht wurde. Denn wir haben PyTorch auf unserer Seite, nicht nur Bleistift und Papier. Selbst wenn Sie mit Ableitungen und Integralen nicht vertraut sind, müssen Sie sich nicht abschrecken lassen.

Diese Themen werde ich so knapp und praktisch wie möglich halten und mich nur auf das konzentrieren, was Ihnen in späteren Kapiteln helfen wird und was unter den Begriff »grundlegende Mathematik« fällt.

Dies ist kein kompletter Crashkurs in Mathematik!

Dieses Buch bietet keineswegs einen umfassenden Überblick über die Mathematik an Gymnasien und Universitäten. Wenn Sie das wollen, empfiehlt sich das Buch *No Bullshit Guide to Math and Physics* von Ivan Savov. Die ersten Kapitel enthalten den besten Crashkurs zur Mathematik auf Abitur- und Universitätsniveau, den ich je gesehen habe. Das Buch *Mathematics 1001* von Dr. Richard Elwes bietet ebenfalls großartige Inhalte und zudem Erklärungen in leicht verdaulichen Häppchen.

Zahlentheorie

Was sind Zahlen? Ich verspreche, in diesem Buch nicht zu philosophisch zu werden, aber sind Zahlen nicht ein Konstrukt, das wir definiert haben? Warum haben wir die Ziffern 0 bis 9 und nicht mehr als diese? Warum haben wir Brüche und Dezimalzahlen und nicht einfach nur ganze Zahlen? Dieser Bereich der Mathematik, in dem wir über Zahlen nachsinnen und warum wir sie in einer bestimmten Weise konstruiert haben, wird als Zahlentheorie bezeichnet.

Die *Zahlentheorie* geht bis in die Antike zurück, als Mathematiker verschiedene Zahlensysteme untersucht haben, und sie erklärt, warum wir sie so akzeptieren, wie wir es heute tun. Hier sind verschiedene Zahlensysteme, die Sie vielleicht kennen:

Natürliche Zahlen
: Das sind die Zahlen 1, 2, 3, 4, 5 ... usw. Zu dieser Klasse gehören nur positive Zahlen. Die natürlichen Zahlen sind das älteste bekannte System, denn bereits die Höhlenmenschen ritzten Striche auf Knochen und Höhlenwände, um Aufzeichnungen zu machen.

Ganze Zahlen
: Zusätzlich zu den natürlichen Zahlen wurde später das Konzept der »0« hinzugefügt. Wir sprechen dann von *ganzen Zahlen*. Die Babylonier entwickelten auch die nützliche Idee, leere »Spalten« für Zahlen größer als 9 freizuhalten, wie zum Beispiel »10«, »1.000« oder »1.090«. Die Nullen zeigen an, dass die Spalten nicht durch einen Wert belegt sind.

Integer-Zahlen
: Integer-Zahlen umfassen sowohl positive als auch negative natürliche Zahlen sowie die 0. Wir mögen sie für selbstverständlich halten, Mathematiker der Antike jedoch misstrauten der Idee von negativen Zahlen zutiefst. Aber wenn man 5 von 3 subtrahiert, erhält man –2. Dies ist vor allem im Finanzwesen nützlich, wenn wir Gewinne und Verluste messen. Im Jahr 628 n. Chr. zeigte ein indischer Mathematiker namens Brahmagupta, warum negative Zahlen notwendig sind, damit die Arithmetik mit der quadratischen Formel möglich wurde. Daher wurden ganze Zahlen – positive und negative – akzeptiert.

Rationale Zahlen
: Jede Zahl, die sich als Bruch ausdrücken lässt, beispielsweise 2/3, ist eine rationale Zahl. Dies schließt alle endlichen Dezimalzahlen und ganzen Zahlen ein, da sie sich auch als Brüche darstellen lassen, wie zum Beispiel 687/100 = 6,87 oder 2/1 = 2. Die Bezeichnung *rationale Zahlen* stammt vom lateinischen Wort *ratio* = *Verhältnis*. Rationale Zahlen wurden schnell als notwendig erachtet, weil Zeit, Ressourcen und andere Größen nicht immer in diskreten Einheiten gemessen werden konnten. Milch wird nicht immer in Litern abgefüllt, wir müssen vielleicht in Teilen eines Liters messen. Wenn ich 12 Minuten laufe, werde ich das nicht in ganzen Kilometern angeben können, da ich bei gemächlichem Tempo eigentlich nur 1,7 Kilometer absolviert habe.

Irrationale Zahlen

Irrationale Zahlen lassen sich nicht als Bruch ausdrücken. Dieser Klasse gehören die berühmte Kreiszahl π an, die Quadratwurzeln einiger Zahlen wie $\sqrt{2}$ und die eulersche Zahl e, die Sie später noch kennenlernen werden. Diese Zahlen haben eine unendliche Anzahl von Dezimalstellen, etwa so: 3,14159265358979323846 2...

Hinter den irrationalen Zahlen verbirgt sich eine interessante Geschichte. Der griechische Mathematiker Pythagoras glaubte, dass alle Zahlen rational seien. Er glaubte so fest daran, dass er eine Religion gründete, die die Zahl 10 anbetete. »Segne uns, göttliche Zahl, du, die du Götter und Menschen hervorgebracht hast!«, beteten er und seine Anhänger (warum »10« für ihn so besonders war, weiß ich nicht). Eine Legende besagt, dass einer seiner Anhänger, Hippasus, bewies, dass nicht alle Zahlen rational sind, indem er einfach die Quadratwurzel aus 2 demonstrierte. Dies brachte Pythagoras' Glaubenssystem schwer durcheinander, und er reagierte, indem er Hippasus im Meer ertränkte.

Unabhängig davon wissen wir heute, dass nicht alle Zahlen rational sind.

Reelle Zahlen

Zu den reellen Zahlen gehören rationale sowie irrationale Zahlen. In der Praxis können Sie bei datenwissenschaftlichen Arbeiten alle Dezimalstellen, mit denen Sie arbeiten, als reelle Zahlen behandeln.

Komplexe und imaginäre Zahlen

Diesem Zahlentyp werden Sie häufiger begegnen, wenn Sie die Quadratwurzel einer negativen Zahl ziehen. Obwohl imaginäre und komplexe Zahlen für bestimmte Problemstellungen von Bedeutung sind, werden wir sie hier meistens meiden.

In der Data Science werden Sie einen Großteil Ihrer Arbeit (wenn nicht die gesamte Arbeit) mit ganzen Zahlen, natürlichen Zahlen, verschiedenen Integer-Typen und reellen Zahlen zu tun haben. Imaginäre Zahlen werden Sie in erweiterten Anwendungsfällen wiederfinden, beispielsweise als Matrixzerlegung, auf die wir in Kapitel 4 zurückkommen.

Komplexe und imaginäre Zahlen

Wenn Sie mehr über imaginäre Zahlen lernen möchten, gibt es eine großartige Playlist auf YouTube mit dem Titel *Imaginary Numbers are Real* (*https://oreil.ly/bvyIq*, englisch mit deutschen Untertiteln).

Reihenfolge der Operationen

Hoffentlich sind Sie mit der *Reihenfolge der Operationen* vertraut, d. h. mit der Reihenfolge, in der Sie die einzelnen Teile eines mathematischen Ausdrucks lösen. Eine kurze Auffrischung: Zuerst werten Sie die Teile in Klammern aus, gefolgt von den Potenzen, dann kommen Multiplikation, Division, Addition und Subtraktion.

Aufgrund der zugeordneten Rechenzeichen werden Multiplikation und Division als Punktrechnung und Addition und Subtraktion als Strichrechnung bezeichnet. Die Reihenfolge der Operationen können Sie sich dann mit der Eselsbrücke *KlaPoPuS* merken, also Klammern vor Potenzen vor Punkt- und Strichrechnungen. Nehmen Sie zum Beispiel folgenden Ausdruck:

$$2 \times \frac{(3+2)^2}{5} - 4$$

Zuerst werten wir den Inhalt der Klammern (3 + 2) aus, was gleich 5 ist:

$$2 \times \frac{(\mathbf{5})^2}{5} - 4$$

Als Nächstes lösen wir die Potenz auf, hier das Quadrat der eben gebildeten Summe 5, also 25:

$$2 \times \frac{\mathbf{25}}{5} - 4$$

Nun sind Multiplikation und Division dran. Die Reihenfolge dieser beiden Operationen lässt sich vertauschen, da Division prinzipiell eine Multiplikation (mit Brüchen) ist. Wir multiplizieren 2 mit $\frac{25}{5}$, was $\frac{50}{5}$ ergibt:

$$\frac{50}{5} - 4$$

Wir führen die Division aus, d.h. 50 durch 5, was 10 ergibt:

$$\mathbf{10} - 4$$

Zum Schluss ist die Strichrechnung an der Reihe, d.h. Addition und Subtraktion. Im Beispiel ist nur noch 10 – 4 zu rechnen, was 6 liefert:

$$10 - 4 = 6$$

In Python ausgedrückt, würden wir natürlich einen Wert von `6.0` erhalten, wie Beispiel 1-1 zeigt.

Beispiel 1-1: Einen Ausdruck in Python lösen

```
my_value = 2 * (3 + 2)**2 / 5 - 4
print(my_value) # Ausgabe: 6.0
```

Dies mag zwar elementar sein, ist aber dennoch von entscheidender Bedeutung. Auch wenn Sie im Code ohne Klammern das richtige Ergebnis erhalten, empfiehlt es sich, Klammern in komplexen Ausdrücken großzügig zu verwenden, damit Sie immer die Kontrolle über die Auswertungsreihenfolge behalten.

In Beispiel 1-2 gruppiere ich den gebrochenen Teil meines Ausdrucks in Klammern, um ihn deutlich vom Rest des Ausdrucks abzugrenzen.

Beispiel 1-2: Mit Klammern in Python für Klarheit sorgen

```
my_value = 2 * ((3 + 2)**2 / 5) - 4

print(my_value) # Ausgabe: 6.0
```

Zwar sind beide Beispiele technisch korrekt, doch ist das zweite für uns leicht zu verwirrende Menschen übersichtlicher. Sollten Sie oder jemand anderes Ihren Code überarbeiten, bieten die Klammern einen gut erkennbaren Verweis auf die Reihenfolge der Operationen, wenn Sie Änderungen vornehmen. Dies bietet zudem eine Verteidigungslinie gegenüber Codeänderungen, um Fehler zu vermeiden.

Variablen

Wenn Sie bereits Skripte in Python oder einer anderen Programmiersprache geschrieben haben, wissen Sie schon in etwa, was eine Variable ist. In der Mathematik versteht man unter einer *Variablen* einen benannten Platzhalter für eine nicht festgelegte oder unbekannte Zahl.

Vielleicht haben Sie eine Variable x, die eine beliebige reelle Zahl darstellt, und Sie können diese Variable multiplizieren, ohne zu deklarieren, was sie ist. In Beispiel 1-3 nehmen wir eine Variableneingabe x von einem Benutzer und multiplizieren sie mit 3.

Beispiel 1-3: Eine Variable in Python, die dann multipliziert wird

```
x = int(input("Bitte eine Zahl eingeben\n"))

product = 3 * x

print(product)
```

Für bestimmte Variablentypen gibt es einige Standardvariablennamen. Wenn Ihnen diese Variablennamen und Konzepte nicht vertraut sind, keine Angst! Einige Leser werden aber erkennen, dass wir Theta (θ) verwenden, um Winkel zu bezeichnen, und Beta (β) für einen Parameter in einer linearen Regression. Griechische Buchstaben ergeben heikle Variablennamen in Python, wie Beispiel 1-4 zeigt.

Beispiel 1-4: Griechische Variablennamen in Python

```
beta = 1.75
theta = 30.0
```

Beachten Sie auch, dass sich Variablennamen indizieren lassen, sodass Sie mehrere Instanzen eines Variablennamens verwenden können. Aus praktischen Gründen behandeln wir diese als eigenständige Variablen. Wenn Sie die Variablen $x1$, $x2$ und $x3$ sehen, nehmen Sie sie einfach als drei separate Variablen, wie Beispiel 1-5 zeigt.

Beispiel 1-5: Indizierte Variablen in Python ausdrücken

```
x1 = 3 # oder x_1 = 3
x2 = 10 # oder x_2 = 10
x3 = 44 # oder x_3 = 44
```

Funktionen

Funktionen sind Ausdrücke, die Beziehungen zwischen zwei oder mehr Variablen definieren. Genauer gesagt, übernimmt eine Funktion *Eingabevariablen* (auch *Domänenvariablen* oder *unabhängige Variablen* genannt), fügt sie in einen Ausdruck ein und liefert dann eine *Ausgabevariable* (auch *abhängige Variable* genannt).

Nehmen Sie diese einfache lineare Funktion:

$$y = 2x + 1$$

Für jeden gegebenen x-Wert lösen wir den Ausdruck mit diesem x auf, um y zu ermitteln. Wenn $x = 1$, dann ist $y = 3$. Wenn $x = 2$, dann ist $y = 5$. Wenn $x = 3$, dann ist $y = 7$ und so weiter, wie Tabelle 1-1 zeigt.

Tabelle 1-1: Verschiedene Werte für $y = 2x + 1$

x	2x + 1	y
0	2(0) + 1	1
1	2(1) + 1	3
2	2(2) + 1	5
3	2(3) + 1	7

Funktionen sind nützlich, weil sie eine vorhersehbare Beziehung zwischen Variablen modellieren, beispielsweise wie viele Brände y bei einer Temperatur x zu erwarten sind. Lineare Funktionen werden wir in Kapitel 5 für lineare Regressionen verwenden.

Eine weitere Konvention besteht für die abhängige Variable y darin, sie explizit als Funktion von x zu bezeichnen, beispielsweise $f(x)$. Anstatt eine Funktion als $y = 2x + 1$ auszudrücken, können wir sie auch wie folgt schreiben:

$$f(x) = 2x + 1$$

Beispiel 1-6 zeigt, wie wir eine mathematische Funktion deklarieren und in Python durchlaufen können.

Beispiel 1-6: Eine lineare Funktion in Python deklarieren

```
def f(x):
  return 2 * x + 1

x_values = [0, 1, 2, 3]

for x in x_values:
  y = f(x)
  print(y)
```

Beim Umgang mit reellen Zahlen ist ein subtiles, aber wichtiges Feature von Funktionen zu beachten: Oftmals haben sie eine unendliche Anzahl von x-Werten und

resultierenden y-Werten. Stellen Sie sich folgende Frage: Wie viele x-Werte können wir durch die Funktion $y = 2x + 1$ darstellen? Anstatt nur 0, 1, 2, 3, ... warum nicht auch 0, 0,5, 1, 1,5, 2, 2,5, 3, wie in Tabelle 1-2 gezeigt?

Tabelle 1-2: Verschiedene Werte für $y = 2x + 1$

x	2x + 1	y
0,0	2(0) + 1	1
0,5	2(0,5) + 1	2
1,0	2(1) + 1	3
1,5	2(1,5) + 1	4
2,0	2(2) + 1	5
2,5	2(2,5) + 1	6
3,0	2(3) + 1	7

Oder warum nicht Viertelschritte für x? Oder Zehntelschritte? Diese Schritte können wir unendlich klein machen und damit zeigen, dass $y = 2x + 1$ eine stetige Funktion ist, bei der es für jeden möglichen Wert von x einen Wert von y gibt. Dies ist eine gute Überleitung, um unsere Funktion als Gerade zu visualisieren, wie Abbildung 1-1 zeigt.

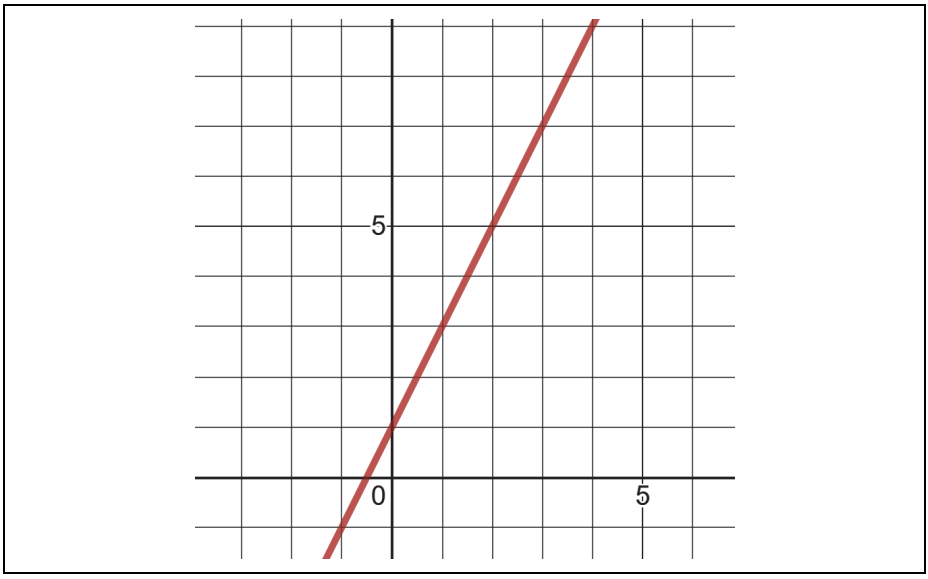

Abbildung 1-1: Graph der Funktion $y = 2x + 1$

Wenn wir auf einer zweidimensionalen Ebene mit zwei Zahlengeraden (einer für jede Variable) zeichnen, sprechen wir von einer *kartesischen Ebene*, *x-y-Ebene* oder *Koordinatenebene*. Wir tragen einen gegebenen x-Wert auf, gehen dann zum entsprechenden y-Wert und zeichnen eine Gerade durch die Schnittpunkte. Beachten

Sie, dass es aufgrund der Natur der reellen Zahlen (oder Dezimalzahlen, wenn Sie das bevorzugen) eine unendliche Anzahl von x-Werten gibt. Deshalb erhalten wir bei der Darstellung der Funktion $f(x)$ eine durchgehende Linie ohne Unterbrechungen. Es gibt eine unendliche Anzahl von Punkten auf dieser Linie oder einem Teil dieser Linie.

Wenn Sie dies mit Python plotten wollen, können Sie auf eine Reihe von Diagrammbibliotheken zurückgreifen – von Plotly bis matplotlib. In diesem Buch ziehen wir SymPy für viele Aufgaben heran, und als erste Aufgabe plotten wir eine Funktion. SymPy stützt sich auf matplotlib, sodass dieses Paket installiert sein muss. Andernfalls erscheint ein hässlicher textbasierter Graph auf der Konsole. Danach deklarieren Sie einfach die Variable x für SymPy mithilfe von `symbols()`, deklarieren Ihre Funktion und plotten sie dann, wie Beispiel 1-7 und Abbildung 1-2 zeigen.

Beispiel 1-7: Eine lineare Funktion in Python mithilfe von SymPy als Graph darstellen

```
from sympy import *

x = symbols('x')
f = 2*x + 1
plot(f)
```

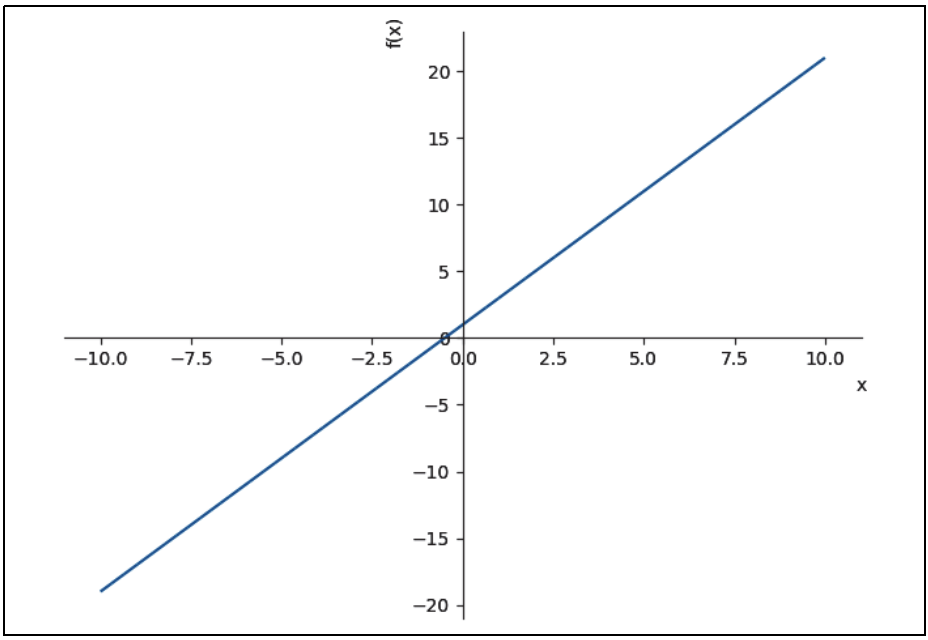

Abbildung 1-2: Mithilfe von SymPy eine lineare Funktion als Graph darstellen

Beispiel 1-8 und Abbildung 1-3 zeigen ein weiteres Beispiel, das die Funktion $f(x) = x^2 + 1$ darstellt.

Beispiel 1-8: Diagrammdarstellung einer Potenzfunktion

```
from sympy import *

x = symbols('x')
f = x**2 + 1
plot(f)
```

Der in Abbildung 1-3 dargestellte Verlauf ist keine Gerade, sondern eine glatte, symmetrische Kurve, eine sogenannte Parabel. Sie ist stetig, aber nicht linear, da sie keine Werte auf einer geraden Linie erzeugt. Gekrümmte Funktionen wie diese sind schwieriger zu handhaben, aber Sie werden einige Tricks lernen, damit es nicht ganz so schlimm wird.

Krummlinige Funktionen
Wenn eine Funktion nicht linear und gerade ist, sondern gekrümmt, sprechen wir von einer *krummlinigen Funktion*.

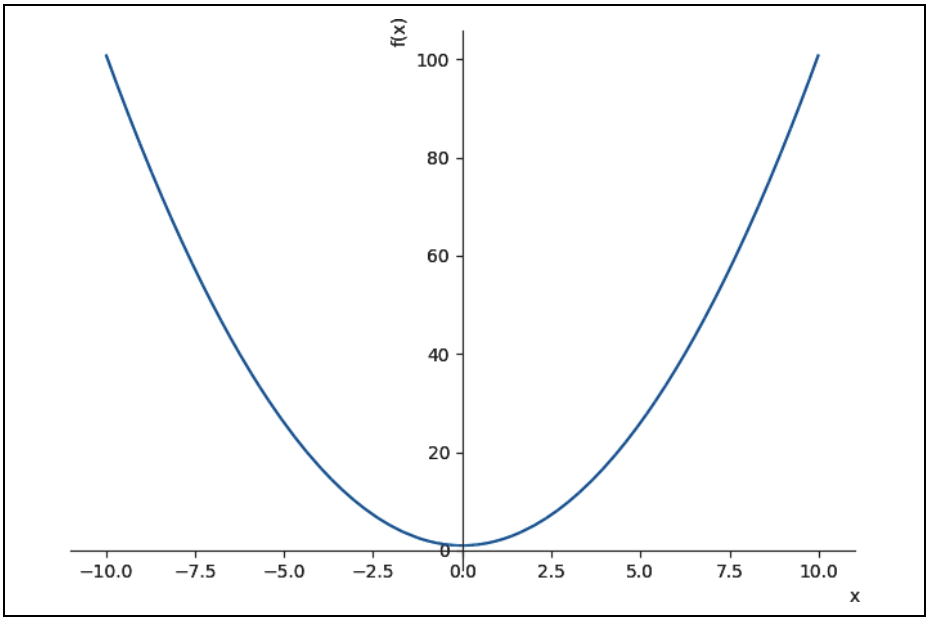

Abbildung 1-3: Eine Potenzfunktion mithilfe von SymPy grafisch darstellen

Funktionen verarbeiten auch mehrere Eingabevariablen, nicht nur eine. Zum Beispiel können wir eine Funktion mit den unabhängigen Variablen x und y schreiben. Hierbei ist y keine abhängige Variable wie in den vorherigen Beispielen.

$$f(x, y) = 2x + 3y$$

Da wir zwei unabhängige Variablen (x und y) und eine abhängige Variable (die Ausgabe von $f(x, y)$) haben, müssen wir diesen Graphen in drei Dimensionen plot-

ten, um eine Ebene von Werten statt einer Linie zu erzeugen, wie in Beispiel 1-9 und Abbildung 1-4 zu sehen.

Beispiel 1-9: Eine Funktion mit zwei unabhängigen Variablen in Python deklarieren

```
from sympy import *
from sympy.plotting import plot3d

x, y = symbols('x y')
f = 2*x + 3*y
plot3d(f)
```

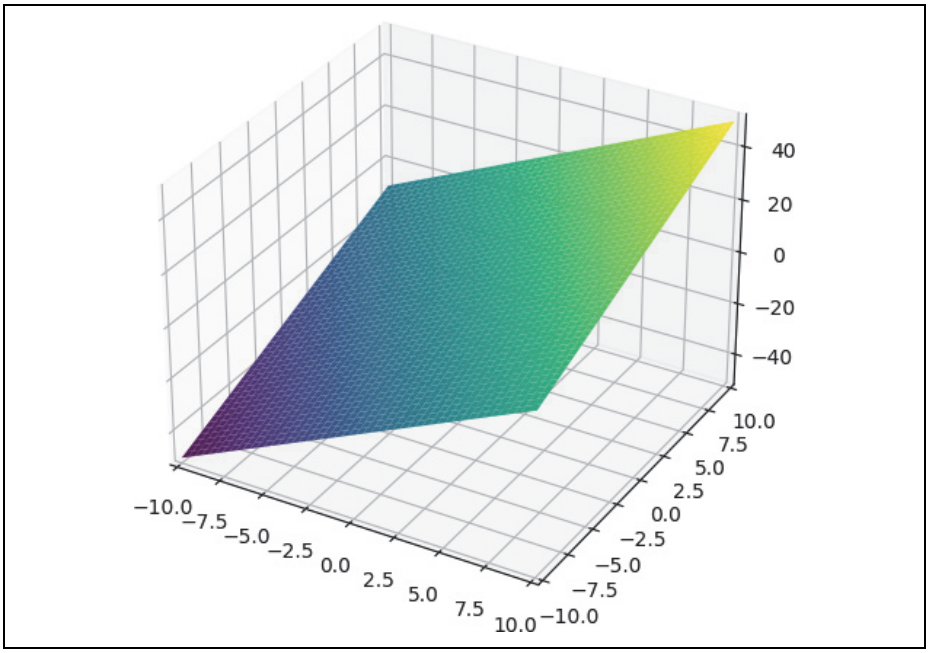

Abbildung 1-4: Eine dreidimensionale Funktion mithilfe von SymPy grafisch darstellen

Unabhängig davon, wie viele unabhängige Variablen Sie haben, wird Ihre Funktion typischerweise nur eine abhängige Variable ausgeben.

Wenn Sie Lösungen für mehrere abhängige Variablen benötigen, werden Sie wahrscheinlich separate Funktionen für die jeweiligen Variablen verwenden.

Summationen

Ich habe Ihnen versprochen, in diesem Buch keine Gleichungen voll mit griechischen Symbolen zu präsentieren. Allerdings gibt es ein Symbol, das so häufig und nützlich ist, dass es nachlässig wäre, es zu übergehen. Eine *Summation* kennzeichnet mit dem großen griechischen Sigma Σ, dass die angegebenen Elemente summiert werden.

Möchte ich zum Beispiel die Zahlen 1 bis 5 durchlaufen, jeweils mit 2 multiplizieren und dann summieren, würde ich das mit einem Summenzeichen wie folgt ausdrücken:

$$\sum_{i=1}^{5} 2i = (2)1 + (2)2 + (2)3 + (2)4 + (2)5 = 30$$

Beispiel 1-10 zeigt, wie sich dies in Python umsetzen lässt.

Beispiel 1-10: Eine Summation in Python ausführen

```
summation = sum(2*i for i in range(1,6))
print(summation)
```

Hier ist i eine Platzhaltervariable für die aufeinanderfolgenden Indexwerte, die wir in der Schleife durchlaufen, jeweils mit 2 multiplizieren und dann aufsummieren. Wenn Sie Daten in einer Schleife verarbeiten, haben Sie es oft mit Variablen wie x_{i} zu tun, die jeweils ein Element in einer Auflistung am Index i kennzeichnet.

Die Funktion range()
Die Funktion `range()` in Python schließt den Endwert aus. Wenn Sie also `range(1,4)` aufrufen, durchläuft die Funktion die Zahlen 1, 2 und 3. Die 4 wird als obere Grenze ausgeschlossen.

Es ist auch üblich, mit n die Anzahl der Elemente in einer Auflistung zu bezeichnen, etwa die Anzahl der Datensätze in einem Dataset. Das folgende Beispiel iteriert über eine Auflistung von Zahlen der Größe n, multipliziert die Zahlen jeweils mit 10 und summiert die Ergebnisse:

$$\sum_{i=1}^{n} 10x_i$$

Der Python-Code in Beispiel 1-11 führt diesen Ausdruck auf einer Auflistung von vier Zahlen aus. Beachten Sie, dass die Elemente in Python (wie in den meisten Programmiersprachen) mit Index 0 beginnend referenziert werden, während wir in der Mathematik bei Index 1 beginnen. Daher verschieben wir unsere Iteration entsprechend, indem wir in unserer Funktion `range()` bei 0 beginnen.

Beispiel 1-11: Summation von Elementen in Python

```
x = [1, 4, 6, 2]
n = len(x)

summation = sum(10*x[i] for i in range(0,n))
print(summation)
```

Das ist das Wesentliche der Summation. Kurz gesagt, drückt eine Summation Σ aus: »Addiere eine Reihe von Elementen und verwende einen Index i sowie einen Maximalwert n, um die einzelnen Iterationen darzustellen, die in die Summe eingehen. Derartige Ausdrücke werden Sie das ganze Buch hindurch finden.

Summationen in SymPy

Auf diesen Kasten können Sie gern später zurückkommen, wenn Sie mehr über SymPy erfahren. SymPy, mit der wir grafische Funktionen darstellen, ist eigentlich eine symbolische mathematische Bibliothek. Was das bedeutet, darüber sprechen wir später in diesem Kapitel. Für die Zukunft können Sie sich zumindest merken, dass in SymPy eine Summenbildung mit dem Operator `Sum()` ausgeführt wird. Im folgenden Code iterieren wir `i` von `1` bis `n`, multiplizieren jeweils mit `i` und summieren dann. Anschließend aber verwenden wir die Funktion `subs()`, um `n` als `5` festzulegen, sodass wir dann alle Elemente von `i` bis `n` summieren.

```
from sympy import *

i,n = symbols('i n')

# jedes Element i von 1 bis n durchlaufen,
# dann multiplizieren und summieren
summation = Sum(2*i,(i,1,n))

# n als 5 festlegen,
# über die Zahlen 1 bis 5 iterieren
up_to_5 = summation.subs(n, 5)
print(up_to_5.doit()) # 30
```

Die Summationen in SymPy sind träge, sodass weder automatische Berechnungen stattfinden noch Ausdrücke vereinfacht werden. Deshalb empfiehlt es sich, den Ausdruck mithilfe der Funktion `doit()` auszuführen.

Potenzen

Potenzen multiplizieren eine Zahl mit sich selbst mit einer festgelegten Anzahl von Malen. Wenn Sie 2 zur dritten Potenz erheben möchten (ausgedrückt als 23 mit dem Exponenten 3), werden drei 2en miteinander multipliziert:

$$2^3 = 2 * 2 * 2 = 8$$

Die *Basis* ist die Variable oder der Wert, den wir potenzieren, und der *Exponent* gibt an, wie oft wir den Basiswert multiplizieren. Für den Ausdruck 2^3 ist 2 die Basis und 3 der Exponent.

Potenzen besitzen einige interessante Eigenschaften. Angenommen, wir multiplizierten x^2 und x^3. Wenn ich nun die Exponenten mit einer einfachen Multiplikation erweitere und dann zu einem einzelnen Exponenten zusammenfasse, sieht das so aus:

$$x^2x^3 = (x * x) * (x * x * x) = x^{2+3} = x^5$$

Wenn wir Potenzen mit der gleichen Basis miteinander multiplizieren, addieren wir einfach die Exponenten, was als *1. Potenzgesetz* bekannt ist. Lassen Sie mich