

Texts in Computational Science and Engineering

6

Editors

Timothy J. Barth
Michael Griebel
David E. Keyes
Risto M. Nieminen
Dirk Roose
Tamar Schlick

More information about this series at <http://www.springer.com/series/5151>

Hans Petter Langtangen

A Primer on Scientific Programming with Python

5th Edition

 Springer

Hans Petter Langtangen
Simula Research Laboratory
Fornebu, Norway

On leave from:

Department of Informatics
University of Oslo
Oslo, Norway

ISSN 1611-0994

Texts in Computational Science and Engineering

ISBN 978-3-662-49886-6

ISBN 978-3-662-49887-3 (eBook)

DOI 10.1007/978-3-662-49887-3

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2016945366

Mathematic Subject Classification to (2010): 26-01, 34A05, 34A30, 34A34, 39-01, 40-01, 65D15, 65D25, 65D30, 68-01, 68N01, 68N19, 68N30, 70-01, 92D25, 97-04, 97U50

© Springer-Verlag Berlin Heidelberg 2009, 2011, 2012, 2014, 2016

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer Berlin Heidelberg

Preface

The aim of this book is to teach computer programming using examples from mathematics and the natural sciences. We have chosen to use the Python programming language because it combines remarkable expressive power with very clean, simple, and compact syntax. Python is easy to learn and very well suited for an introduction to computer programming. Python is also quite similar to MATLAB and a good language for doing mathematical computing. It is easy to combine Python with compiled languages, like Fortran, C, and C++, which are widely used languages for scientific computations.

The examples in this book integrate programming with applications to mathematics, physics, biology, and finance. The reader is expected to have knowledge of basic one-variable calculus as taught in mathematics-intensive programs in high schools. It is certainly an advantage to take a university calculus course in parallel, preferably containing both classical and numerical aspects of calculus. Although not strictly required, a background in high school physics makes many of the examples more meaningful.

Many introductory programming books are quite compact and focus on listing functionality of a programming language. However, learning to program is learning how to *think* as a programmer. This book has its main focus on the thinking process, or equivalently: programming as a problem solving technique. That is why most of the pages are devoted to case studies in programming, where we define a problem and explain how to create the corresponding program. New constructions and programming styles (what we could call theory) is also usually introduced via examples. Particular attention is paid to verification of programs and to finding errors. These topics are very demanding for mathematical software, because the unavoidable numerical approximation errors are possibly mixed with programming mistakes.

By studying the many examples in the book, I hope readers will learn how to think right and thereby write programs in a quicker and more reliable way. Remember, nobody can learn programming by just reading – one has to solve a large amount of exercises hands on. The book is therefore full of exercises of various types: modifications of existing examples, completely new problems, or debugging of given programs.

To work with this book, I recommend using Python version 2.7. For Chaps. 5–9 and Appendices A–E, you need the NumPy and Matplotlib packages, preferably

also the IPython and SciTools packages, and for Appendix G, Cython is required. Other packages used in the text are nose and sympy. Section H.1 has more information on how you can get access to Python and the mentioned packages.

There is a web page associated with this book, <http://hplgit.github.io/scipro-primer>, containing all the example programs from the book as well as information on installation of the software on various platforms.

Python version 2 or 3? A common problem among Python programmers is to choose between version 2 or 3, which at the time of this writing means choosing between version 2.7 and 3.5. A common recommendation is to go for Python 3, because this is the version that will be further developed in the future. However, there is a problem that much useful mathematical software in Python has not yet been ported to Python 3. Therefore, Python version 2.7 is the most popular version for doing scientific computing, and that is why also this book applies version 2.7.

A widely used strategy for software developers who want to write Python code that works with both versions, is to develop a common version for Python 2 and 3. For the programs in this book, a common version can easily be produced by first developing for version 2.7 and then *automatically* convert the code by running the `futurize` program. Section 4.10 demonstrates how this is done in simple cases.

The Python 2.7 code in this book sticks to all modern constructions that are backported from version 3 such that the code becomes as close as possible to the equivalent Python 3 code. At any time, you can just run `futurize` to see the differences between your Python 2.7 version and the corresponding Python 3.5 version.

Contents Chapter 1 introduces variables, objects, modules, and text formatting through examples concerning evaluation of mathematical formulas. Chapter 2 presents programming with `while` and `for` loops as well as lists, including nested lists. The next chapter deals with two other fundamental concepts in programming: functions and `if-else` tests.

How to read data into programs and deal with errors in input are the subjects of Chap. 4. Chapter 5 introduces arrays and array computing (including vectorization) and how this is used for plotting $y = f(x)$ curves and making animation of curves. Many of the examples in the first five chapters are strongly related. Typically, formulas from the first chapter are used to produce tables of numbers in the second chapter. Then the formulas are encapsulated in functions in the third chapter. In the next chapter, the input to the functions are fetched from the command line, and validity checks of the input are added. The formulas are then shown as graphs in Chap. 5. After having studied Chaps. 1–5, the reader should have enough knowledge of programming to solve mathematical problems by what many refer to as “MATLAB-style” programming.

Chapter 6 explains how to work with dictionaries and strings, especially for interpreting text data in files and storing the extracted information in flexible data structures. Class programming, including user-defined types for mathematical computations (with overloaded operators), is introduced in Chap. 7. Chapter 8 deals with random numbers and statistical computing with applications to games and random walks. Object-oriented programming, in the meaning of class hierarchies and inheritance, is the subject of Chap. 9. The key examples here deal with building toolkits for numerical differentiation and integration as well as graphics.

Appendix A introduces mathematical modeling, using sequences and difference equations. Only programming concepts from Chaps. 1–5 are used in this appendix, the aim being to consolidate basic programming knowledge and apply it to mathematical problems. Some important mathematical topics are introduced via difference equations in a simple way: Newton’s method, Taylor series, inverse functions, and dynamical systems.

Appendix B deals with functions on a mesh, numerical differentiation, and numerical integration. A simple introduction to ordinary differential equations and their numerical treatment is provided in Appendix C. Appendix D shows how a complete project in physics can be solved by mathematical modeling, numerical methods, and programming elements from Chaps. 1–5. This project is a good example on problem solving in computational science, where it is necessary to integrate physics, mathematics, numerics, and computer science.

How to create software for solving ordinary differential equations, using both function-based and object-oriented programming, is the subject of Appendix E. The material in this appendix brings together many parts of the book in the context of physical applications and differential equations.

Appendix F is devoted to the art of debugging, and in fact problem solving in general. Speeding up numerical computations in Python by migrating code to C via Cython is exemplified in Appendix G. Finally, Appendix H deals with various more advanced technical topics.

Most of the examples and exercises in this book are quite short. However, many of the exercises are related, and together they form larger projects, for example on Fourier Series (3.21, 4.21, 4.22, 5.41, 5.42), numerical integration (3.11, 3.12, 5.49, 5.50, A.12), Taylor series (3.37, 5.32, 5.39, A.14, A.15, 7.23), piecewise constant functions (3.29–3.33, 5.34, 5.47, 5.48, 7.19–7.21), inverse functions (E.17–E.20), falling objects (E.8, E.9, E.38, E.39), oscillatory population growth (A.19, A.21, A.22, A.23), epidemic disease modeling (E.41–E.48), optimization and finance (A.24, 8.42, 8.43), statistics and probability (4.24, 4.25, 8.23, 8.24), hazard games (8.8–8.14), random walk and statistical physics (8.32–8.40), noisy data analysis (8.44–8.46), numerical methods (5.25–5.27, 7.8, 7.9, A.9, 7.22, 9.15–9.17, E.30–E.37), building a calculus calculator (7.34, 9.18, 9.19), and creating a toolkit for simulating vibrating engineering systems (E.50–E.55).

Chapters 1–9 together with Appendices A and E have from 2007 formed the core of an introductory first semester bachelor course on scientific programming at the University of Oslo (INF1100, 10 ECTS credits).

Changes from the fourth to the fifth edition Substantial changes were introduced in the fourth edition, and the fifth edition is primarily a consolidation of those changes. Many typos have been corrected and many explanations and exercises have been improved. The emphasis on unit tests and test functions, especially in exercises, is stronger than in the previous edition. Symbolic computation with the aid of SymPy is used to a larger extent and integrated with numerical computing throughout the book. All classes are now new-style (instead of old-style/classic as in previous editions). Examples on Matplotlib do not use the pylab module anymore, but pyplot and MATLAB-like syntax is still favored to ease the transition between Python and MATLAB. The concept of closures is more explicit than in earlier editions (see the new Sect. 7.1.7) since this is a handy and popular construc-

tion much used in the scientific Python community. We also discuss the difference between Python 2 and 3 and demonstrate how to use the `future` module to write code that runs under both versions.

The most substantial new material in the fifth edition appears toward the end of Chap. 5 and regards high-performance computing, linear algebra, and visualization of scalar and vector fields. Although this material is not used elsewhere in the book, many readers have requested basic recipes when going from one to two variables or from vectors to matrices later when solving more advanced problems and using the book as their programming reference. The new material in Chap. 5 was written jointly with Dr. Øyvind Ryan.

Acknowledgments This book was born out of stimulating discussions with my close colleague Aslak Tveito, and he started writing what is now Appendix B and C. The whole book project and the associated university course were critically dependent on Aslak's enthusiastic role back in 2007. The continuous support from Aslak regarding my book projects is much appreciated and contributes greatly to my strong motivation. Another key contributor in the early days was Ilmar Wilbers. He made extensive efforts with assisting the book project and establishing the university course INF1100. I feel that without Ilmar and his solutions to numerous technical problems the first edition of the book would never have been completed. Johannes H. Ring also deserves special acknowledgment for the development of the Easyviz graphics tool back in the days when Python plotting was a hassle, and later for his maintenance of software associated with the book.

Professor Loyce Adams studied the entire book, solved all the exercises, found numerous errors, and suggested many improvements. Her contributions are so much appreciated. More recently, Helmut Büch worked extremely carefully through all details in Chaps. 1–6, tested the software, found many typos, and asked critical questions that led to lots of significant improvements. I am so thankful for all his efforts and for his enthusiasm during the preparations of the fourth edition. The fifth edition has benefited much from Hakki Eres' careful examination of the fourth edition. He found several typos and code errors, some of which go back to the first edition.

Special thanks go to Geir Kjetil Sandve for being the primary author of the computational bioinformatics examples in Sects. 3.3, 6.5, 8.3.4, and 9.5, with contributions from Sveinung Gundersen, Ksenia Khelik, Halfdan Rydbeck, and Kai Trengereid. I am also grateful to Øyvind Ryan's work with linear algebra and visualization of scalar and vector fields in Chap. 5.

Several people have contributed with suggestions for improvements of the text, the exercises, and the associated software. I will in particular mention Ingrid Eide, Ståle Zerener Haugnæss, Kristian Hiorth, Timothy Keough, Arve Knudsen, Espen Kristensen, Tobias Vidarssønn Langhoff, Martin Vonheim Larsen, Kine Veronica Lund, Solveig Masvie, Håkon Møller, Rebekka Mørken, Mathias Nedrebø, Marit Sandstad, Helene Norheim Semmerud, Lars Storjord, Fredrik Heffer Valdmanis, and Torkil Vederhus. Hakon Adler is greatly acknowledged for his careful reading of early various versions of the manuscript. Many thanks go to the professors Fred Espen Bent, Ørnulf Borgan, Geir Dahl, Knut Mørken, and Geir Pedersen for formulating several exciting exercises from various application fields. I also appreciate the cover image made by my good friend Jan Olav Langseth.

This book and the associated course are parts of a comprehensive and successful reform at the University of Oslo, called *Computing in Science Education*. The goal of the reform is to integrate computer programming and simulation in all bachelor courses in natural science where mathematical models are used. The present book lays the foundation for the modern computerized problem solving technique to be applied in later courses. It has been extremely inspiring to work closely with the driving forces behind this reform, especially the professors Morten Hjorth-Jensen, Anders Malthe-Sørenssen, Knut Mørken, and Arnt Inge Vistnes.

The excellent assistance from the Springer system over the years, in particular Martin Peters, Thanh-Ha Le Thi, Ruth Allewelt, Peggy Glauch-Ruge, Nadja Kroke, Thomas Schmidt, Patrick Waltemate, Donatas Akmanavicius, and Yvonne Schlatter, is highly appreciated, and ensured a smooth and rapid production of all editions of this book.

Oslo, February 2016

Hans Petter Langtangen

Contents

1	Computing with Formulas	1
1.1	The First Programming Encounter: a Formula	1
1.1.1	Using a Program as a Calculator	2
1.1.2	About Programs and Programming	2
1.1.3	Tools for Writing Programs	3
1.1.4	Writing and Running Your First Python Program	4
1.1.5	Warning About Typing Program Text	5
1.1.6	Verifying the Result	6
1.1.7	Using Variables	6
1.1.8	Names of Variables	6
1.1.9	Reserved Words in Python	7
1.1.10	Comments	8
1.1.11	Formatting Text and Numbers	9
1.2	Computer Science Glossary	12
1.3	Another Formula: Celsius-Fahrenheit Conversion	16
1.3.1	Potential Error: Integer Division	16
1.3.2	Objects in Python	17
1.3.3	Avoiding Integer Division	18
1.3.4	Arithmetic Operators and Precedence	20
1.4	Evaluating Standard Mathematical Functions	20
1.4.1	Example: Using the Square Root Function	20
1.4.2	Example: Computing with $\sinh x$	23
1.4.3	A First Glimpse of Rounding Errors	23
1.5	Interactive Computing	24
1.5.1	Using the Python Shell	25
1.5.2	Type Conversion	26
1.5.3	IPython	27
1.6	Complex Numbers	29
1.6.1	Complex Arithmetics in Python	30
1.6.2	Complex Functions in Python	31
1.6.3	Unified Treatment of Complex and Real Functions	31
1.7	Symbolic Computing	33
1.7.1	Basic Differentiation and Integration	33
1.7.2	Equation Solving	34

1.7.3	Taylor Series and More	35
1.8	Summary	35
1.8.1	Chapter Topics	35
1.8.2	Example: Trajectory of a Ball	39
1.8.3	About Typesetting Conventions in This Book	40
1.9	Exercises	41
2	Loops and Lists	51
2.1	While Loops	51
2.1.1	A Naive Solution	51
2.1.2	While Loops	52
2.1.3	Boolean Expressions	54
2.1.4	Loop Implementation of a Sum	56
2.2	Lists	57
2.2.1	Basic List Operations	57
2.2.2	For Loops	60
2.3	Alternative Implementations with Lists and Loops	62
2.3.1	While Loop Implementation of a for Loop	62
2.3.2	The Range Construction	63
2.3.3	For Loops with List Indices	64
2.3.4	Changing List Elements	65
2.3.5	List Comprehension	66
2.3.6	Traversing Multiple Lists Simultaneously	66
2.4	Nested Lists	67
2.4.1	A table as a List of Rows or Columns	67
2.4.2	Printing Objects	68
2.4.3	Extracting Sublists	70
2.4.4	Traversing Nested Lists	72
2.5	Tuples	74
2.6	Summary	75
2.6.1	Chapter Topics	75
2.6.2	Example: Analyzing List Data	78
2.6.3	How to Find More Python Information	80
2.7	Exercises	82
3	Functions and Branching	91
3.1	Functions	91
3.1.1	Mathematical Functions as Python Functions	91
3.1.2	Understanding the Program Flow	93
3.1.3	Local and Global Variables	94
3.1.4	Multiple Arguments	96
3.1.5	Function Argument or Global Variable?	97
3.1.6	Beyond Mathematical Functions	98
3.1.7	Multiple Return Values	99
3.1.8	Computing Sums	100
3.1.9	Functions with No Return Values	101
3.1.10	Keyword Arguments	103
3.1.11	Doc Strings	105

3.1.12	Functions as Arguments to Functions	107
3.1.13	The Main Program	109
3.1.14	Lambda Functions	110
3.2	Branching	110
3.2.1	If-else Blocks	111
3.2.2	Inline if Tests	113
3.3	Mixing Loops, Branching, and Functions in Bioinformatics	
Examples		113
3.3.1	Counting Letters in DNA Strings	114
3.3.2	Efficiency Assessment	118
3.3.3	Verifying the Implementations	120
3.4	Summary	121
3.4.1	Chapter Topics	121
3.4.2	Example: Numerical Integration	123
3.5	Exercises	127
4	User Input and Error Handling	149
4.1	Asking Questions and Reading Answers	150
4.1.1	Reading Keyboard Input	150
4.2	Reading from the Command Line	151
4.2.1	Providing Input on the Command Line	151
4.2.2	A Variable Number of Command-Line Arguments	152
4.2.3	More on Command-Line Arguments	153
4.3	Turning User Text into Live Objects	154
4.3.1	The Magic Eval Function	154
4.3.2	The Magic Exec Function	158
4.3.3	Turning String Expressions into Functions	160
4.4	Option-Value Pairs on the Command Line	161
4.4.1	Basic Usage of the Argparse Module	162
4.4.2	Mathematical Expressions as Values	163
4.5	Reading Data from File	165
4.5.1	Reading a File Line by Line	166
4.5.2	Alternative Ways of Reading a File	167
4.5.3	Reading a Mixture of Text and Numbers	169
4.6	Writing Data to File	171
4.6.1	Example: Writing a Table to File	171
4.6.2	Standard Input and Output as File Objects	173
4.6.3	What is a File, Really?	176
4.7	Handling Errors	179
4.7.1	Exception Handling	180
4.7.2	Raising Exceptions	183
4.8	A Glimpse of Graphical User Interfaces	185
4.9	Making Modules	188
4.9.1	Example: Interest on Bank Deposits	188
4.9.2	Collecting Functions in a Module File	189
4.9.3	Test Block	190
4.9.4	Verification of the Module Code	192
4.9.5	Getting Input Data	193

4.9.6	Doc Strings in Modules	195
4.9.7	Using Modules	196
4.9.8	Distributing Modules	199
4.9.9	Making Software Available on the Internet	200
4.10	Making Code for Python 2 and 3	201
4.10.1	Basic Differences Between Python 2 and 3	201
4.10.2	Turning Python 2 Code into Python 3 Code	202
4.11	Summary	204
4.11.1	Chapter Topics	204
4.11.2	Example: Bisection Root Finding	208
4.12	Exercises	216
5	Array Computing and Curve Plotting	227
5.1	Vectors	228
5.1.1	The Vector Concept	228
5.1.2	Mathematical Operations on Vectors	229
5.1.3	Vector Arithmetics and Vector Functions	231
5.2	Arrays in Python Programs	232
5.2.1	Using Lists for Collecting Function Data	232
5.2.2	Basics of Numerical Python Arrays	233
5.2.3	Computing Coordinates and Function Values	235
5.2.4	Vectorization	236
5.3	Curve Plotting	238
5.3.1	MATLAB-Style Plotting with Matplotlib	238
5.3.2	Matplotlib; Pyplot Prefix	243
5.3.3	SciTools and Easyviz	244
5.3.4	Making Animations	249
5.3.5	Making Videos	254
5.3.6	Curve Plots in Pure Text	255
5.4	Plotting Difficulties	256
5.4.1	Piecewisely Defined Functions	256
5.4.2	Rapidly Varying Functions	259
5.5	More Advanced Vectorization of Functions	260
5.5.1	Vectorization of StringFunction Objects	260
5.5.2	Vectorization of the Heaviside Function	261
5.5.3	Vectorization of a Hat Function	265
5.6	More on Numerical Python Arrays	267
5.6.1	Copying Arrays	267
5.6.2	In-Place Arithmetics	268
5.6.3	Allocating Arrays	269
5.6.4	Generalized Indexing	269
5.6.5	Testing for the Array Type	270
5.6.6	Compact Syntax for Array Generation	271
5.6.7	Shape Manipulation	271
5.7	High-Performance Computing with Arrays	272
5.7.1	Scalar Implementation	272
5.7.2	Vectorized Implementation	273
5.7.3	Memory-Saving Implementation	273

5.7.4	Analysis of Memory Usage	275
5.7.5	Analysis of the CPU Time	276
5.8	Higher-Dimensional Arrays	277
5.8.1	Matrices and Arrays	277
5.8.2	Two-Dimensional Numerical Python Arrays	278
5.8.3	Array Computing	281
5.8.4	Matrix Objects	282
5.9	Some Common Linear Algebra Operations	283
5.9.1	Inverse, Determinant, and Eigenvalues	283
5.9.2	Products	283
5.9.3	Norms	284
5.9.4	Sum and Extreme Values	284
5.9.5	Indexing	286
5.9.6	Transpose and Upper/Lower Triangular Parts	286
5.9.7	Solving Linear Systems	287
5.9.8	Matrix Row and Column Operations	287
5.9.9	Computing the Rank of a Matrix	288
5.9.10	Symbolic Linear Algebra	289
5.10	Plotting of Scalar and Vector Fields	292
5.10.1	Installation	292
5.10.2	Surface Plots	293
5.10.3	Parameterized Curve	293
5.10.4	Contour Lines	294
5.10.5	The Gradient Vector Field	294
5.11	Matplotlib	296
5.11.1	Surface Plots	296
5.11.2	Contour Plots	297
5.11.3	Vector Field Plots	299
5.12	Mayavi	299
5.12.1	Surface Plots	300
5.12.2	Contour Plots	303
5.12.3	Vector Field Plots	303
5.12.4	A 3D Scalar Field and Its Gradient Field	304
5.12.5	Animations	306
5.13	Summary	307
5.13.1	Chapter Topics	307
5.13.2	Example: Animating a Function	308
5.14	Exercises	313
6	Dictionaries and Strings	333
6.1	Dictionaries	333
6.1.1	Making Dictionaries	334
6.1.2	Dictionary Operations	334
6.1.3	Example: Polynomials as Dictionaries	336
6.1.4	Dictionaries with Default Values and Ordering	338
6.1.5	Example: Storing File Data in Dictionaries	341
6.1.6	Example: Storing File Data in Nested Dictionaries	342

6.1.7	Example: Reading and Plotting Data Recorded at Specific Dates	347
6.2	Strings	351
6.2.1	Common Operations on Strings	351
6.2.2	Example: Reading Pairs of Numbers	355
6.2.3	Example: Reading Coordinates	358
6.3	Reading Data from Web Pages	360
6.3.1	About Web Pages	361
6.3.2	How to Access Web Pages in Programs	362
6.3.3	Example: Reading Pure Text Files	363
6.3.4	Example: Extracting Data from HTML	365
6.3.5	Handling Non-English Text	366
6.4	Reading and Writing Spreadsheet Files	369
6.4.1	CSV Files	369
6.4.2	Reading CSV Files	370
6.4.3	Processing Spreadsheet Data	371
6.4.4	Writing CSV Files	372
6.4.5	Representing Number Cells with Numerical Python Arrays	373
6.4.6	Using More High-Level Numerical Python Functionality	374
6.5	Examples from Analyzing DNA	375
6.5.1	Computing Frequencies	375
6.5.2	Analyzing the Frequency Matrix	382
6.5.3	Finding Base Frequencies	385
6.5.4	Translating Genes into Proteins	388
6.5.5	Some Humans Can Drink Milk, While Others Cannot	393
6.6	Making Code that is Compatible with Python 2 and 3	394
6.6.1	More Basic Differences Between Python 2 and 3	394
6.6.2	Turning Python 2 Code into Python 3 Code	396
6.7	Summary	396
6.7.1	Chapter Topics	396
6.7.2	Example: A File Database	398
6.8	Exercises	402
7	Introduction to Classes	409
7.1	Simple Function Classes	409
7.1.1	Challenge: Functions with Parameters	410
7.1.2	Representing a Function as a Class	412
7.1.3	The Self Variable	417
7.1.4	Another Function Class Example	419
7.1.5	Alternative Function Class Implementations	420
7.1.6	Making Classes Without the Class Construct	422
7.1.7	Closures	424
7.2	More Examples on Classes	426
7.2.1	Bank Accounts	426
7.2.2	Phone Book	428
7.2.3	A Circle	430
7.3	Special Methods	432
7.3.1	The Call Special Method	432

7.3.2	Example: Automagic Differentiation	433
7.3.3	Example: Automagic Integration	438
7.3.4	Turning an Instance into a String	440
7.3.5	Example: Phone Book with Special Methods	441
7.3.6	Adding Objects	443
7.3.7	Example: Class for Polynomials	443
7.3.8	Arithmetic Operations and Other Special Methods	449
7.3.9	Special Methods for String Conversion	449
7.4	Example: Class for Vectors in the Plane	451
7.4.1	Some Mathematical Operations on Vectors	451
7.4.2	Implementation	452
7.4.3	Usage	454
7.5	Example: Class for Complex Numbers	455
7.5.1	Implementation	455
7.5.2	Illegal Operations	457
7.5.3	Mixing Complex and Real Numbers	457
7.5.4	Dynamic, Static, Strong, Weak, and Duck Typing	459
7.5.5	Special Methods for “Right” Operands	460
7.5.6	Inspecting Instances	461
7.6	Static Methods and Attributes	463
7.7	Summary	464
7.7.1	Chapter Topics	464
7.7.2	Example: Interval Arithmetic	466
7.8	Exercises	470
8	Random Numbers and Simple Games	489
8.1	Drawing Random Numbers	489
8.1.1	The Seed	490
8.1.2	Uniformly Distributed Random Numbers	491
8.1.3	Visualizing the Distribution	492
8.1.4	Vectorized Drawing of Random Numbers	493
8.1.5	Computing the Mean and Standard Deviation	494
8.1.6	The Gaussian or Normal Distribution	496
8.2	Drawing Integers	497
8.2.1	Random Integer Functions	498
8.2.2	Example: Throwing a Die	498
8.2.3	Drawing a Random Element from a List	501
8.2.4	Example: Drawing Cards from a Deck	502
8.2.5	Example: Class Implementation of a Deck	504
8.3	Computing Probabilities	507
8.3.1	Principles of Monte Carlo Simulation	507
8.3.2	Example: Throwing Dice	508
8.3.3	Example: Drawing Balls from a Hat	511
8.3.4	Random Mutations of Genes	513
8.3.5	Example: Policies for Limiting Population Growth	519
8.4	Simple Games	522
8.4.1	Guessing a Number	522
8.4.2	Rolling Two Dice	523

8.5	Monte Carlo Integration	526
8.5.1	Derivation of Monte Carlo Integration	526
8.5.2	Implementation of Standard Monte Carlo Integration	528
8.5.3	Area Computing by Throwing Random Points	531
8.6	Random Walk in One Space Dimension	534
8.6.1	Basic Implementation	534
8.6.2	Visualization	535
8.6.3	Random Walk as a Difference Equation	536
8.6.4	Computing Statistics of the Particle Positions	536
8.6.5	Vectorized Implementation	537
8.7	Random Walk in Two Space Dimensions	539
8.7.1	Basic Implementation	539
8.7.2	Vectorized Implementation	541
8.8	Summary	542
8.8.1	Chapter Topics	542
8.8.2	Example: Random Growth	544
8.9	Exercises	549
9	Object-Oriented Programming	567
9.1	Inheritance and Class Hierarchies	567
9.1.1	A Class for Straight Lines	568
9.1.2	A First Try on a Class for Parabolas	569
9.1.3	A Class for Parabolas Using Inheritance	569
9.1.4	Checking the Class Type	571
9.1.5	Attribute vs Inheritance: has-a vs is-a Relationship	572
9.1.6	Superclass for Defining an Interface	574
9.2	Class Hierarchy for Numerical Differentiation	576
9.2.1	Classes for Differentiation	577
9.2.2	Verification	579
9.2.3	A flexible Main Program	581
9.2.4	Extensions	582
9.2.5	Alternative Implementation via Functions	585
9.2.6	Alternative Implementation via Functional Programming	586
9.2.7	Alternative Implementation via a Single Class	587
9.3	Class Hierarchy for Numerical Integration	589
9.3.1	Numerical Integration Methods	589
9.3.2	Classes for Integration	590
9.3.3	Verification	594
9.3.4	Using the Class Hierarchy	595
9.3.5	About Object-Oriented Programming	597
9.4	Class Hierarchy for Making Drawings	599
9.4.1	Using the Object Collection	600
9.4.2	Example of Classes for Geometric Objects	609
9.4.3	Adding Functionality via Recursion	614
9.4.4	Scaling, Translating, and Rotating a Figure	618
9.5	Classes for DNA Analysis	620
9.5.1	Class for Regions	620
9.5.2	Class for Genes	621

- 9.5.3 Subclasses 626
- 9.6 Summary 627
 - 9.6.1 Chapter Topics 627
 - 9.6.2 Example: Input Data Reader 629
- 9.7 Exercises 635

- A Sequences and Difference Equations 645**
 - A.1 Mathematical Models Based on Difference Equations 646
 - A.1.1 Interest Rates 647
 - A.1.2 The Factorial as a Difference Equation 649
 - A.1.3 Fibonacci Numbers 650
 - A.1.4 Growth of a Population 651
 - A.1.5 Logistic Growth 652
 - A.1.6 Payback of a Loan 654
 - A.1.7 The Integral as a Difference Equation 655
 - A.1.8 Taylor Series as a Difference Equation 657
 - A.1.9 Making a Living from a Fortune 658
 - A.1.10 Newton’s Method 659
 - A.1.11 The Inverse of a Function 663
 - A.2 Programming with Sound 665
 - A.2.1 Writing Sound to File 666
 - A.2.2 Reading Sound from File 667
 - A.2.3 Playing Many Notes 667
 - A.2.4 Music of a Sequence 668
 - A.3 Exercises 671

- B Introduction to Discrete Calculus 683**
 - B.1 Discrete Functions 683
 - B.1.1 The Sine Function 684
 - B.1.2 Interpolation 685
 - B.1.3 Evaluating the Approximation 686
 - B.1.4 Generalization 687
 - B.2 Differentiation Becomes Finite Differences 688
 - B.2.1 Differentiating the Sine Function 689
 - B.2.2 Differences on a Mesh 690
 - B.2.3 Generalization 692
 - B.3 Integration Becomes Summation 693
 - B.3.1 Dividing into Subintervals 693
 - B.3.2 Integration on Subintervals 695
 - B.3.3 Adding the Subintervals 696
 - B.3.4 Generalization 697
 - B.4 Taylor Series 699
 - B.4.1 Approximating Functions Close to One Point 699
 - B.4.2 Approximating the Exponential Function 699
 - B.4.3 More Accurate Expansions 700
 - B.4.4 Accuracy of the Approximation 702
 - B.4.5 Derivatives Revisited 704
 - B.4.6 More Accurate Difference Approximations 705

	B.4.7 Second-Order Derivatives	707
	B.5 Exercises	709
C	Introduction to differential equations	715
	C.1 The simplest case	716
	C.2 Exponential Growth	718
	C.3 Logistic Growth	723
	C.4 A Simple Pendulum	724
	C.5 A Model for the Spreading of a Disease	727
	C.6 Exercises	729
D	A Complete Differential Equation Project	731
	D.1 About the Problem: Motion and Forces in Physics	731
	D.1.1 The Physical Problem	731
	D.1.2 The Computational Algorithm	733
	D.1.3 Derivation of the Mathematical Model	734
	D.1.4 Derivation of the Algorithm	736
	D.2 Program Development and Testing	737
	D.2.1 Implementation	737
	D.2.2 Callback Functionality	740
	D.2.3 Making a Module	742
	D.2.4 Verification	743
	D.3 Visualization	746
	D.3.1 Simultaneous Computation and Plotting	746
	D.3.2 Some Applications	748
	D.3.3 Remark on Choosing Δt	749
	D.3.4 Comparing Several Quantities in Subplots	750
	D.3.5 Comparing Approximate and Exact Solutions	751
	D.3.6 Evolution of the Error as Δt Decreases	752
	D.4 Exercises	755
E	Programming of Differential Equations	757
	E.1 Scalar Ordinary Differential Equations	758
	E.1.1 Examples on Right-Hand-Side Functions	758
	E.1.2 The Forward Euler Scheme	759
	E.1.3 Function Implementation	760
	E.1.4 Verifying the Implementation	761
	E.1.5 From Discrete to Continuous Solution	763
	E.1.6 Switching Numerical Method	764
	E.1.7 Class Implementation	764
	E.1.8 Logistic Growth via a Function-Based Approach	769
	E.1.9 Logistic Growth via a Class-Based Approach	769
	E.2 Systems of Ordinary Differential Equations	772
	E.2.1 Mathematical Problem	773
	E.2.2 Example of a System of ODEs	774
	E.2.3 Function Implementation	775
	E.2.4 Class Implementation	777
	E.3 The ODESolver Class Hierarchy	779

E.3.1	Numerical Methods	779
E.3.2	Construction of a Solver Hierarchy	780
E.3.3	The Backward Euler Method	783
E.3.4	Verification	785
E.3.5	Example: Exponential Decay	787
E.3.6	Example: The Logistic Equation with Problem and Solver Classes	789
E.3.7	Example: An Oscillating System	797
E.3.8	Application 4: The Trajectory of a Ball	799
E.3.9	Further Developments of ODESolver	801
E.4	Exercises	802
F	Debugging	835
F.1	Using a Debugger	835
F.2	How to Debug	838
F.2.1	A Recipe for Program Writing and Debugging	838
F.2.2	Application of the Recipe	841
F.2.3	Getting Help from a Code Analyzer	853
G	Migrating Python to Compiled Code	857
G.1	Pure Python Code for Monte Carlo Simulation	857
G.1.1	The Computational Problem	858
G.1.2	A Scalar Python Implementation	858
G.1.3	A Vectorized Python Implementation	859
G.2	Migrating Scalar Python Code to Cython	860
G.2.1	A Plain Cython Implementation	860
G.2.2	A Better Cython Implementation	863
G.3	Migrating Code to C	865
G.3.1	Writing a C Program	865
G.3.2	Migrating Loops to C Code via F2PY	866
G.3.3	Migrating Loops to C Code via Cython	867
G.3.4	Comparing Efficiency	868
H	Technical Topics	871
H.1	Getting Access to Python	871
H.1.1	Required Software	871
H.1.2	Installing Software on Your Laptop: Mac OS X and Windows	872
H.1.3	Anaconda and Spyder	873
H.1.4	VMWare Fusion Virtual Machine	874
H.1.5	Dual Boot on Windows	877
H.1.6	Vagrant Virtual Machine	877
H.2	How to Write and Run a Python Program	878
H.2.1	The Need for a Text Editor	878
H.2.2	Terminal Windows	880
H.3	The SageMathCloud and Wakari Web Services	880
H.3.1	Basic Intro to SageMathCloud	880
H.3.2	Basic Intro to Wakari	881

H.3.3	Installing Your Own Python Packages	881
H.4	Writing IPython Notebooks	882
H.4.1	A Simple Program in the Notebook	882
H.4.2	Mixing Text, Mathematics, Code, and Graphics	882
H.5	Different Ways of Running Python Programs	884
H.5.1	Executing Python Programs in iPython	884
H.5.2	Executing Python Programs in Unix	884
H.5.3	Executing Python Programs in Windows	885
H.5.4	Executing Python Programs in Mac OS X	887
H.5.5	Making a Complete Stand-Alone Executable	887
H.6	Doing Operating System Tasks in Python	888
H.7	Variable Number of Function Arguments	891
H.7.1	Variable Number of Positional Arguments	891
H.7.2	Variable Number of Keyword Arguments	894
H.8	Evaluating Program Efficiency	896
H.8.1	Making Time Measurements	896
H.8.2	Profiling Python Programs	898
H.9	Software Testing	899
H.9.1	Requirements of the Test Function	900
H.9.2	Writing the Test Function; Precomputed Data	900
H.9.3	Writing the Test Function; Exact Numerical Solution	901
H.9.4	Testing of Function Robustness	902
H.9.5	Automatic Execution of Tests	904
	References	907
	Index	909

List of Exercises

Exercise 1.1: Compute 1+1	42
Exercise 1.2: Write a Hello World program	43
Exercise 1.3: Derive and compute a formula	43
Exercise 1.4: Convert from meters to British length units	43
Exercise 1.5: Compute the mass of various substances	43
Exercise 1.6: Compute the growth of money in a bank	43
Exercise 1.7: Find error(s) in a program	43
Exercise 1.8: Type in program text	43
Exercise 1.9: Type in programs and debug them	44
Exercise 1.10: Evaluate a Gaussian function	45
Exercise 1.11: Compute the air resistance on a football	45
Exercise 1.12: How to cook the perfect egg	46
Exercise 1.13: Derive the trajectory of a ball	46
Exercise 1.14: Find errors in the coding of formulas	47
Exercise 1.15: Explain why a program does not work	47
Exercise 1.16: Find errors in Python statements	48
Exercise 1.17: Find errors in the coding of a formula	48
Exercise 1.18: Find errors in a program	49
Exercise 2.1: Make a Fahrenheit-Celsius conversion table	82
Exercise 2.2: Generate an approximate Fahrenheit-Celsius conversion table	82
Exercise 2.3: Work with a list	82
Exercise 2.4: Generate odd numbers	82
Exercise 2.5: Compute the sum of the first n integers	82
Exercise 2.6: Compute energy levels in an atom	82
Exercise 2.7: Generate equally spaced coordinates	83
Exercise 2.8: Make a table of values from a formula	83
Exercise 2.9: Store values from a formula in lists	83
Exercise 2.10: Simulate operations on lists by hand	84
Exercise 2.11: Compute a mathematical sum	84
Exercise 2.12: Replace a while loop by a for loop	84
Exercise 2.13: Simulate a program by hand	85
Exercise 2.14: Explore Python documentation	85
Exercise 2.15: Index a nested list	85
Exercise 2.16: Store data in lists	86

Exercise 2.17: Store data in a nested list	86
Exercise 2.18: Values of boolean expressions	86
Exercise 2.19: Explore round-off errors from a large number of inverse operations	87
Exercise 2.20: Explore what zero can be on a computer	87
Exercise 2.21: Compare two real numbers with a tolerance	87
Exercise 2.22: Interpret a code	88
Exercise 2.23: Explore problems with inaccurate indentation	88
Exercise 2.24: Explore punctuation in Python programs	89
Exercise 2.25: Investigate a for loop over a changing list	89
Exercise 3.1: Implement a simple mathematical function	127
Exercise 3.2: Implement a simple mathematical function with a parameter	127
Exercise 3.3: Explain how a program works	128
Exercise 3.4: Write a Fahrenheit-Celsius conversion functions	128
Exercise 3.5: Write a test function for Exercise 3.4	128
Exercise 3.6: Given a test function, write the function	128
Exercise 3.7: Evaluate a sum and write a test function	129
Exercise 3.8: Write a function for solving $ax^2 + bx + c = 0$	129
Exercise 3.9: Implement the sum function	129
Exercise 3.10: Compute a polynomial via a product	130
Exercise 3.11: Integrate a function by the Trapezoidal rule	130
Exercise 3.12: Derive the general Midpoint integration rule	131
Exercise 3.13: Make an adaptive Trapezoidal rule	132
Exercise 3.14: Simulate a program by hand	133
Exercise 3.15: Debug a given test function	134
Exercise 3.16: Compute the area of an arbitrary triangle	134
Exercise 3.17: Compute the length of a path	135
Exercise 3.18: Approximate π	135
Exercise 3.19: Compute the area of a polygon	135
Exercise 3.20: Write functions	136
Exercise 3.21: Approximate a function by a sum of sines	136
Exercise 3.22: Implement a Gaussian function	137
Exercise 3.23: Wrap a formula in a function	137
Exercise 3.24: Write a function for numerical differentiation	137
Exercise 3.25: Implement the factorial function	137
Exercise 3.26: Compute velocity and acceleration from 1D position data	138
Exercise 3.27: Find the max and min values of a function	138
Exercise 3.28: Find the max and min elements in a list	139
Exercise 3.29: Implement the Heaviside function	139
Exercise 3.30: Implement a smoothed Heaviside function	139
Exercise 3.31: Implement an indicator function	140
Exercise 3.32: Implement a piecewise constant function	140
Exercise 3.33: Apply indicator functions	141
Exercise 3.34: Test your understanding of branching	141
Exercise 3.35: Simulate nested loops by hand	141
Exercise 3.36: Rewrite a mathematical function	142
Exercise 3.37: Make a table for approximations of $\cos x$	142
Exercise 3.38: Use None in keyword arguments	143

Exercise 3.39: Write a sort function for a list of 4-tuples	143
Exercise 3.40: Find prime numbers	145
Exercise 3.41: Find pairs of characters	145
Exercise 3.42: Count substrings	145
Exercise 3.43: Resolve a problem with a function	145
Exercise 3.44: Determine the types of some objects	146
Exercise 3.45: Find an error in a program	146
Exercise 4.1: Make an interactive program	216
Exercise 4.2: Read a number from the command line	216
Exercise 4.3: Read a number from a file	216
Exercise 4.4: Read and write several numbers from and to file	217
Exercise 4.5: Use exceptions to handle wrong input	217
Exercise 4.6: Read input from the keyboard	217
Exercise 4.7: Read input from the command line	217
Exercise 4.8: Try MSWord or LibreOffice to write a program	218
Exercise 4.9: Prompt the user for input to a formula	218
Exercise 4.10: Read parameters in a formula from the command line	218
Exercise 4.11: Use exceptions to handle wrong input	218
Exercise 4.12: Test validity of input data	219
Exercise 4.13: Raise an exception in case of wrong input	219
Exercise 4.14: Evaluate a formula for data in a file	219
Exercise 4.15: Write a function given its test function	219
Exercise 4.16: Compute the distance it takes to stop a car	220
Exercise 4.17: Look up calendar functionality	221
Exercise 4.18: Use the StringFunction tool	221
Exercise 4.19: Why we test for specific exception types	221
Exercise 4.20: Make a complete module	221
Exercise 4.21: Organize a previous program as a module	222
Exercise 4.22: Read options and values from the command line	222
Exercise 4.23: Check if mathematical identities hold	222
Exercise 4.24: Compute probabilities with the binomial distribution	224
Exercise 4.25: Compute probabilities with the Poisson distribution	224
Exercise 5.1: Fill lists with function values	313
Exercise 5.2: Fill arrays; loop version	313
Exercise 5.3: Fill arrays; vectorized version	313
Exercise 5.4: Plot a function	313
Exercise 5.5: Apply a function to a vector	313
Exercise 5.6: Simulate by hand a vectorized expression	313
Exercise 5.7: Demonstrate array slicing	314
Exercise 5.8: Replace list operations by array computing	314
Exercise 5.9: Plot a formula	314
Exercise 5.10: Plot a formula for several parameters	314
Exercise 5.11: Specify the extent of the axes in a plot	314
Exercise 5.12: Plot exact and inexact Fahrenheit-Celsius conversion formulas	314
Exercise 5.13: Plot the trajectory of a ball	314
Exercise 5.14: Plot data in a two-column file	315
Exercise 5.15: Write function data to file	315
Exercise 5.16: Plot data from a file	316

Exercise 5.17: Write table to file	316
Exercise 5.18: Fit a polynomial to data points	317
Exercise 5.19: Fit a polynomial to experimental data	318
Exercise 5.20: Read acceleration data and find velocities	318
Exercise 5.21: Read acceleration data and plot velocities	319
Exercise 5.22: Plot a trip's path and velocity from GPS coordinates	319
Exercise 5.23: Vectorize the Midpoint rule for integration	320
Exercise 5.24: Vectorize a function for computing the area of a polygon	320
Exercise 5.25: Implement Lagrange's interpolation formula	321
Exercise 5.26: Plot Lagrange's interpolating polynomial	321
Exercise 5.27: Investigate the behavior of Lagrange's interpolating polynomials	322
Exercise 5.28: Plot a wave packet	322
Exercise 5.29: Judge a plot	322
Exercise 5.30: Plot the viscosity of water	323
Exercise 5.31: Explore a complicated function graphically	323
Exercise 5.32: Plot Taylor polynomial approximations to $\sin x$	323
Exercise 5.33: Animate a wave packet	324
Exercise 5.34: Animate a smoothed Heaviside function	324
Exercise 5.35: Animate two-scale temperature variations	324
Exercise 5.36: Use non-uniformly distributed coordinates for visualization	325
Exercise 5.37: Animate a sequence of approximations to π	325
Exercise 5.38: Animate a planet's orbit	325
Exercise 5.39: Animate the evolution of Taylor polynomials	326
Exercise 5.40: Plot the velocity profile for pipeflow	327
Exercise 5.41: Plot sum-of-sines approximations to a function	327
Exercise 5.42: Animate the evolution of a sum-of-sine approximation to a function	327
Exercise 5.43: Plot functions from the command line	328
Exercise 5.44: Improve command-line input	328
Exercise 5.45: Demonstrate energy concepts from physics	328
Exercise 5.46: Plot a w-like function	328
Exercise 5.47: Plot a piecewise constant function	329
Exercise 5.48: Vectorize a piecewise constant function	329
Exercise 5.49: Visualize approximations in the Midpoint integration rule	329
Exercise 5.50: Visualize approximations in the Trapezoidal integration rule	330
Exercise 5.51: Experience overflow in a function	330
Exercise 5.52: Apply a function to a rank 2 array	331
Exercise 5.53: Explain why array computations fail	331
Exercise 5.54: Verify linear algebra results	331
Exercise 6.1: Make a dictionary from a table	402
Exercise 6.2: Explore syntax differences: lists vs. dicts	402
Exercise 6.3: Use string operations to improve a program	403
Exercise 6.4: Interpret output from a program	403
Exercise 6.5: Make a dictionary	403
Exercise 6.6: Make a nested dictionary	404
Exercise 6.7: Make a nested dictionary from a file	404
Exercise 6.8: Make a nested dictionary from a file	404
Exercise 6.9: Compute the area of a triangle	405

Exercise 6.10: Compare data structures for polynomials	405
Exercise 6.11: Compute the derivative of a polynomial	405
Exercise 6.12: Specify functions on the command line	405
Exercise 6.13: Interpret function specifications	406
Exercise 6.14: Compare average temperatures in cities	407
Exercise 6.15: Generate an HTML report with figures	407
Exercise 6.16: Allow different types for a function argument	408
Exercise 6.17: Make a function more robust	408
Exercise 6.18: Find proportion of bases inside/outside exons	408
Exercise 7.1: Make a function class	470
Exercise 7.2: Add a data attribute to a class	471
Exercise 7.3: Add functionality to a class	471
Exercise 7.4: Make classes for a rectangle and a triangle	471
Exercise 7.5: Make a class for quadratic functions	472
Exercise 7.6: Make a class for straight lines	472
Exercise 7.7: Flexible handling of function arguments	472
Exercise 7.8: Wrap functions in a class	473
Exercise 7.9: Flexible handling of function arguments	473
Exercise 7.10: Deduce a class implementation	474
Exercise 7.11: Implement special methods in a class	474
Exercise 7.12: Make a class for summation of series	474
Exercise 7.13: Apply a numerical differentiation class	475
Exercise 7.14: Implement an addition operator	475
Exercise 7.15: Implement in-place += and -= operators	476
Exercise 7.16: Implement a class for numerical differentiation	476
Exercise 7.17: Examine a program	477
Exercise 7.18: Modify a class for numerical differentiation	477
Exercise 7.19: Make a class for the Heaviside function	478
Exercise 7.20: Make a class for the indicator function	478
Exercise 7.21: Make a class for piecewise constant functions	479
Exercise 7.22: Speed up repeated integral calculations	479
Exercise 7.23: Apply a class for polynomials	480
Exercise 7.24: Find a bug in a class for polynomials	480
Exercise 7.25: Implement subtraction of polynomials	480
Exercise 7.26: Test the functionality of pretty print of polynomials	480
Exercise 7.27: Vectorize a class for polynomials	481
Exercise 7.28: Use a dict to hold polynomial coefficients	481
Exercise 7.29: Extend class Vec2D to work with lists/tuples	482
Exercise 7.30: Extend class Vec2D to 3D vectors	483
Exercise 7.31: Use NumPy arrays in class Vec2D	483
Exercise 7.32: Impreciseness of interval arithmetics	484
Exercise 7.33: Make classes for students and courses	484
Exercise 7.34: Find local and global extrema of a function	484
Exercise 7.35: Find the optimal production for a company	485
Exercise 8.1: Flip a coin times	549
Exercise 8.2: Compute a probability	550
Exercise 8.3: Choose random colors	550
Exercise 8.4: Draw balls from a hat	550

Exercise 8.5: Computing probabilities of rolling dice	550
Exercise 8.6: Estimate the probability in a dice game	550
Exercise 8.7: Compute the probability of hands of cards	551
Exercise 8.8: Decide if a dice game is fair	551
Exercise 8.9: Adjust a game to make it fair	551
Exercise 8.10: Make a test function for Monte Carlo simulation	551
Exercise 8.11: Generalize a game	551
Exercise 8.12: Compare two playing strategies	552
Exercise 8.13: Investigate strategies in a game	552
Exercise 8.14: Investigate the winning chances of some games	552
Exercise 8.15: Compute probabilities of throwing two dice	553
Exercise 8.16: Vectorize flipping a coin	553
Exercise 8.17: Vectorize a probability computation	553
Exercise 8.18: Throw dice and compute a small probability	553
Exercise 8.19: Is democracy reliable as a decision maker?	553
Exercise 8.20: Difference equation for random numbers	555
Exercise 8.21: Make a class for drawing balls from a hat	555
Exercise 8.22: Independent versus dependent random numbers	555
Exercise 8.23: Compute the probability of flipping a coin	556
Exercise 8.24: Simulate binomial experiments	557
Exercise 8.25: Simulate a poker game	557
Exercise 8.26: Estimate growth in a simulation model	557
Exercise 8.27: Investigate guessing strategies	557
Exercise 8.28: Vectorize a dice game	558
Exercise 8.29: Compute π by a Monte Carlo method	558
Exercise 8.30: Compute π by a Monte Carlo method	558
Exercise 8.31: Compute π by a random sum	558
Exercise 8.32: 1D random walk with drift	558
Exercise 8.33: 1D random walk until a point is hit	558
Exercise 8.34: Simulate making a fortune from gaming	559
Exercise 8.35: Simulate pollen movements as a 2D random walk	559
Exercise 8.36: Make classes for 2D random walk	559
Exercise 8.37: 2D random walk with walls; scalar version	560
Exercise 8.38: 2D random walk with walls; vectorized version	560
Exercise 8.39: Simulate mixing of gas molecules	561
Exercise 8.40: Simulate slow mixing of gas molecules	561
Exercise 8.41: Guess beer brands	561
Exercise 8.42: Simulate stock prices	562
Exercise 8.43: Compute with option prices in finance	562
Exercise 8.44: Differentiate noise measurements	563
Exercise 8.45: Differentiate noisy signals	564
Exercise 8.46: Model noise in a time signal	565
Exercise 8.47: Speed up Markov chain mutation	566
Exercise 9.1: Demonstrate the magic of inheritance	635
Exercise 9.2: Make polynomial subclasses of parabolas	635
Exercise 9.3: Implement a class for a function as a subclass	636
Exercise 9.4: Create an alternative class hierarchy for polynomials	636
Exercise 9.5: Make circle a subclass of an ellipse	636

Exercise 9.6: Make super- and subclass for a point	636
Exercise 9.7: Modify a function class by subclassing	637
Exercise 9.8: Explore the accuracy of difference formulas	637
Exercise 9.9: Implement a subclass	637
Exercise 9.10: Make classes for numerical differentiation	638
Exercise 9.11: Implement a new subclass for differentiation	638
Exercise 9.12: Understand if a class can be used recursively	638
Exercise 9.13: Represent people by a class hierarchy	638
Exercise 9.14: Add a new class in a class hierarchy	639
Exercise 9.15: Compute convergence rates of numerical integration methods	640
Exercise 9.16: Add common functionality in a class hierarchy	641
Exercise 9.17: Make a class hierarchy for root finding	641
Exercise 9.18: Make a calculus calculator class	641
Exercise 9.19: Compute inverse functions	642
Exercise 9.20: Make line drawing of a person; program	643
Exercise 9.21: Make line drawing of a person; class	643
Exercise 9.22: Animate a person with waving hands	643
Exercise A.1: Determine the limit of a sequence	671
Exercise A.2: Compute π via sequences	672
Exercise A.3: Reduce memory usage of difference equations	672
Exercise A.4: Compute the development of a loan	672
Exercise A.5: Solve a system of difference equations	672
Exercise A.6: Modify a model for fortune development	672
Exercise A.7: Change index in a difference equation	673
Exercise A.8: Construct time points from dates	673
Exercise A.9: Visualize the convergence of Newton's method	674
Exercise A.10: Implement the secant method	674
Exercise A.11: Test different methods for root finding	675
Exercise A.12: Make difference equations for the Midpoint rule	675
Exercise A.13: Compute the arc length of a curve	675
Exercise A.14: Find difference equations for computing $\sin x$	676
Exercise A.15: Find difference equations for computing $\cos x$	677
Exercise A.16: Make a guitar-like sound	677
Exercise A.17: Damp the bass in a sound file	677
Exercise A.18: Damp the treble in a sound file	678
Exercise A.19: Demonstrate oscillatory solutions of the logistic equation	678
Exercise A.20: Automate computer experiments	679
Exercise A.21: Generate an HTML report	679
Exercise A.22: Use a class to archive and report experiments	680
Exercise A.23: Explore logistic growth interactively	681
Exercise A.24: Simulate the price of wheat	681
Exercise B.1: Interpolate a discrete function	709
Exercise B.2: Study a function for different parameter values	710
Exercise B.3: Study a function and its derivative	710
Exercise B.4: Use the Trapezoidal method	711
Exercise B.5: Compute a sequence of integrals	711
Exercise B.6: Use the Trapezoidal method	712
Exercise B.7: Compute trigonometric integrals	712

Exercise B.8: Plot functions and their derivatives	713
Exercise B.9: Use the Trapezoidal method	714
Exercise C.1: Solve a nonhomogeneous linear ODE	729
Exercise C.2: Solve a nonlinear ODE	729
Exercise C.3: Solve an ODE for $y(x)$	729
Exercise C.4: Experience instability of an ODE	730
Exercise C.5: Solve an ODE with time-varying growth	730
Exercise D.1: Model sudden movements of the plate	755
Exercise D.2: Write a callback function	756
Exercise D.3: Improve input to the simulation program	756
Exercise E.1: Solve a simple ODE with function-based code	802
Exercise E.2: Solve a simple ODE with class-based code	802
Exercise E.3: Solve a simple ODE with the ODEsolver hierarchy	802
Exercise E.4: Solve an ODE specified on the command line	802
Exercise E.5: Implement a numerical method for ODEs	803
Exercise E.6: Solve an ODE for emptying a tank	803
Exercise E.7: Solve an ODE for the arc length	804
Exercise E.8: Simulate a falling or rising body in a fluid	804
Exercise E.9: Verify the limit of a solution as time grows	805
Exercise E.10: Scale the logistic equation	806
Exercise E.11: Compute logistic growth with time-varying carrying capacity	806
Exercise E.12: Solve an ODE until constant solution	807
Exercise E.13: Use a problem class to hold data about an ODE	807
Exercise E.14: Derive and solve a scaled ODE problem	808
Exercise E.15: Clean up a file to make it a module	809
Exercise E.16: Simulate radioactive decay	809
Exercise E.17: Compute inverse functions by solving an ODE	809
Exercise E.18: Make a class for computing inverse functions	810
Exercise E.19: Add functionality to a class	810
Exercise E.20: Compute inverse functions by interpolation	811
Exercise E.21: Code the 4th-order Runge-Kutta method; function	811
Exercise E.22: Code the 4th-order Runge-Kutta method; class	811
Exercise E.23: Compare ODE methods	811
Exercise E.24: Code a test function for systems of ODEs	812
Exercise E.25: Code Heun's method for ODE systems; function	812
Exercise E.26: Code Heun's method for ODE systems; class	812
Exercise E.27: Implement and test the Leapfrog method	812
Exercise E.28: Implement and test an Adams-Bashforth method	813
Exercise E.29: Solve two coupled ODEs for radioactive decay	813
Exercise E.30: Implement a 2nd-order Runge-Kutta method; function	813
Exercise E.31: Implement a 2nd-order Runge-Kutta method; class	813
Exercise E.32: Code the iterated midpoint method; function	814
Exercise E.33: Code the iterated midpoint method; class	814
Exercise E.34: Make a subclass for the iterated midpoint method	815
Exercise E.35: Compare the accuracy of various methods for ODEs	815
Exercise E.36: Animate how various methods for ODEs converge	815
Exercise E.37: Study convergence of numerical methods for ODEs	815
Exercise E.38: Find a body's position along with its velocity	816

Exercise E.39: Add the effect of air resistance on a ball	816
Exercise E.40: Solve an ODE system for an electric circuit	817
Exercise E.41: Simulate the spreading of a disease by a SIR model	817
Exercise E.42: Introduce problem and solver classes in the SIR model	819
Exercise E.43: Introduce vaccination in a SIR model	820
Exercise E.44: Introduce a vaccination campaign in a SIR model	820
Exercise E.45: Find an optimal vaccination period	821
Exercise E.46: Simulate human-zombie interaction	821
Exercise E.47: Simulate a zombie movie	823
Exercise E.48: Simulate a war on zombies	824
Exercise E.49: Explore predator-prey population interactions	824
Exercise E.50: Formulate a 2nd-order ODE as a system	825
Exercise E.51: Solve $\ddot{u} + u = 0$	826
Exercise E.52: Make a tool for analyzing oscillatory solutions	826
Exercise E.53: Implement problem, solver, and visualizer classes	827
Exercise E.54: Use classes for flexible choices of models	831
Exercise E.55: Apply software for oscillating systems	831
Exercise E.56: Model the economy of fishing	832