Python for Software Design

*Python for Software Design* is a concise introduction to software design using the Python programming language. Intended for people with no programming experience, this book starts with the most basic concepts and gradually adds new material. Some of the ideas students find most challenging, like recursion and object-oriented programming, are divided into a sequence of smaller steps and introduced over the course of several chapters. The focus is on the programming process, with special emphasis on debugging. The book includes a wide range of exercises, from short examples to substantial projects, so that students have ample opportunity to practice each new concept.

Exercise solutions and code examples along with Swampy, a suite of Python programs that is used in some of the exercises, are available from thinkpython.com.

Allen B. Downey, Ph.D., is an Associate Professor of Computer Science at the Olin College of Engineering in Needham, Massachusetts. He has taught at Wellesley College, Colby College, and UC Berkeley. He has a doctorate in computer science from UC Berkeley and a master’s degree from MIT. Professor Downey is the author of a previous version of this book, titled *How to Think Like a Computer Scientist: Learning with Python*, which he self-published in 2001.
CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore,
São Paulo, Delhi

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org

Information on this title: www.cambridge.org/9780521725965

© Allen B. Downey 2009

This publication is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without the written
permission of Cambridge University Press.

First published 2009

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication data
Downey, Allen.
Python for software design: how to think like a computer scientist / Allen B. Downey.
p. cm.
Includes index.
1. Python (Computer program language) I. Title.
QA76.73.P98D693 2009
005.13’.0 –dc22 2008054459


Cambridge University Press has no responsibility for the persistence or
accuracy of URLs for external or third-party Internet Web sites referred to in
this publication and does not guarantee that any content on such Web sites is,
or will remain, accurate or appropriate. Information regarding prices, travel
timetables, and other factual information given in this work are correct at
the time of first printing, but Cambridge University Press does not guarantee
the accuracy of such information thereafter.
Contents

Preface xi

1 The Way of the Program 1
   1.1 The Python Programming Language 1
   1.2 What Is a Program? 3
   1.3 What Is Debugging? 3
      1.3.1 Syntax Errors 3
      1.3.2 Runtime Errors 4
      1.3.3 Semantic Errors 4
      1.3.4 Experimental Debugging 4
   1.4 Formal and Natural Languages 5
   1.5 The First Program 6
   1.6 Debugging 7
   1.7 Glossary 8
   1.8 Exercises 9

2 Variables, Expressions, and Statements 10
   2.1 Values and Types 10
   2.2 Variables 11
   2.3 Variable Names and Keywords 13
   2.4 Statements 13
   2.5 Operators and Operands 14
   2.6 Expressions 15
   2.7 Order of Operations 15
   2.8 String Operations 16
   2.9 Comments 17
   2.10 Debugging 17
   2.11 Glossary 18
   2.12 Exercises 19
## Contents

### 3 Functions 21

#### 3.1 Function Calls 21
#### 3.2 Type Conversion Functions 21
#### 3.3 Math Functions 22
#### 3.4 Composition 23
#### 3.5 Adding New Functions 24
#### 3.6 Definitions and Uses 26
#### 3.7 Flow of Execution 26
#### 3.8 Parameters and Arguments 27
#### 3.9 Variables and Parameters Are Local 28
#### 3.10 Stack Diagrams 29
#### 3.11 Fruitful Functions and Void Functions 30
#### 3.12 Why Functions? 31
#### 3.13 Debugging 31
#### 3.14 Glossary 32
#### 3.15 Exercises 33

### 4 Case Study: Interface Design 35

#### 4.1 TurtleWorld 35
#### 4.2 Simple Repetition 36
#### 4.3 Exercises 37
#### 4.4 Encapsulation 38
#### 4.5 Generalization 39
#### 4.6 Interface Design 40
#### 4.7 Refactoring 41
#### 4.8 A Development Plan 42
#### 4.9 Docstring 43
#### 4.10 Debugging 43
#### 4.11 Glossary 44
#### 4.12 Exercises 44

### 5 Conditionals and Recursion 46

#### 5.1 Modulus Operator 46
#### 5.2 Boolean Expressions 46
#### 5.3 Logical Operators 47
#### 5.4 Conditional Execution 48
#### 5.5 Alternative Execution 48
#### 5.6 Chained Conditionals 49
#### 5.7 Nested Conditionals 49
#### 5.8 Recursion 50
#### 5.9 Stack Diagrams for Recursive Functions 52
#### 5.10 Infinite Recursion 52
#### 5.11 Keyboard Input 53
#### 5.12 Debugging 54
#### 5.13 Glossary 55
#### 5.14 Exercises 56
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>6  Fruitful Functions 59</td>
</tr>
<tr>
<td>6.1 Return Values 59</td>
</tr>
<tr>
<td>6.2 Incremental Development 60</td>
</tr>
<tr>
<td>6.3 Composition 63</td>
</tr>
<tr>
<td>6.4 Boolean Functions 64</td>
</tr>
<tr>
<td>6.5 More Recursion 65</td>
</tr>
<tr>
<td>6.6 Leap of Faith 67</td>
</tr>
<tr>
<td>6.7 One More Example 67</td>
</tr>
<tr>
<td>6.8 Checking Types 68</td>
</tr>
<tr>
<td>6.9 Debugging 69</td>
</tr>
<tr>
<td>6.10 Glossary 70</td>
</tr>
<tr>
<td>6.11 Exercises 71</td>
</tr>
<tr>
<td>7  Iteration 73</td>
</tr>
<tr>
<td>7.1 Multiple Assignment 73</td>
</tr>
<tr>
<td>7.2 Updating Variables 74</td>
</tr>
<tr>
<td>7.3 The <code>while</code> Statement 75</td>
</tr>
<tr>
<td>7.4 <code>break</code> 76</td>
</tr>
<tr>
<td>7.5 Square Roots 77</td>
</tr>
<tr>
<td>7.6 Algorithms 79</td>
</tr>
<tr>
<td>7.7 Debugging 79</td>
</tr>
<tr>
<td>7.8 Glossary 80</td>
</tr>
<tr>
<td>7.9 Exercises 80</td>
</tr>
<tr>
<td>8  Strings 82</td>
</tr>
<tr>
<td>8.1 A String Is a Sequence 82</td>
</tr>
<tr>
<td>8.2 <code>len</code> 83</td>
</tr>
<tr>
<td>8.3 Traversal with a <code>for</code> Loop 83</td>
</tr>
<tr>
<td>8.4 String Slices 85</td>
</tr>
<tr>
<td>8.5 Strings Are Immutable 86</td>
</tr>
<tr>
<td>8.6 Searching 86</td>
</tr>
<tr>
<td>8.7 Looping and Counting 87</td>
</tr>
<tr>
<td>8.8 <code>string</code> Methods 87</td>
</tr>
<tr>
<td>8.9 The <code>in</code> Operator 89</td>
</tr>
<tr>
<td>8.10 String Comparison 89</td>
</tr>
<tr>
<td>8.11 Debugging 90</td>
</tr>
<tr>
<td>8.12 Glossary 92</td>
</tr>
<tr>
<td>8.13 Exercises 92</td>
</tr>
<tr>
<td>9  Case Study: Word Play 95</td>
</tr>
<tr>
<td>9.1 Reading Word Lists 95</td>
</tr>
<tr>
<td>9.2 Exercises 96</td>
</tr>
<tr>
<td>9.3 Search 97</td>
</tr>
<tr>
<td>9.4 Looping with Indices 99</td>
</tr>
<tr>
<td>9.5 Debugging 100</td>
</tr>
<tr>
<td>9.6 Glossary 101</td>
</tr>
<tr>
<td>9.7 Exercises 101</td>
</tr>
</tbody>
</table>
# Contents

## 10 Lists 103

10.1 A List Is a Sequence 103  
10.2 Lists Are Mutable 104  
10.3 Traversing a List 105  
10.4 List Operations 106  
10.5 List Slices 106  
10.6 List Methods 107  
10.7 Map, Filter, and Reduce 108  
10.8 Deleting Elements 109  
10.9 Lists and Strings 110  
10.10 Objects and Values 111  
10.11 Aliasing 113  
10.12 List Arguments 113  
10.13 Debugging 115  
10.14 Glossary 116  
10.15 Exercises 117

## 11 Dictionaries 119

11.1 Dictionary as a Set of Counters 121  
11.2 Looping and Dictionaries 123  
11.3 Reverse Lookup 123  
11.4 Dictionaries and Lists 124  
11.5 Memos 126  
11.6 Global Variables 128  
11.7 Long Integers 129  
11.8 Debugging 130  
11.9 Glossary 131  
11.10 Exercises 131

## 12 Tuples 133

12.1 Tuples Are Immutable 133  
12.2 Tuple Assignment 135  
12.3 Tuples as Return Values 136  
12.4 Variable-Length Argument Tuples 136  
12.5 Lists and Tuples 138  
12.6 Dictionaries and Tuples 139  
12.7 Comparing Tuples 141  
12.8 Sequences of Sequences 142  
12.9 Debugging 143  
12.10 Glossary 144  
12.11 Exercises 145

## 13 Case Study: Data Structure Selection 147

13.1 Word Frequency Analysis 147  
13.2 Random Numbers 148  
13.3 Word Histogram 149  
13.4 Most Common Words 151
Contents

13.5 Optional Parameters 152
13.6 Dictionary Subtraction 152
13.7 Random Words 153
13.8 Markov Analysis 154
13.9 Data Structures 155
13.10 Debugging 157
13.11 Glossary 158
13.12 Exercises 158

14 Files 159
14.1 Persistence 159
14.2 Reading and Writing 159
14.3 Format Operator 160
14.4 Filenames and Paths 161
14.5 Catching Exceptions 163
14.6 Databases 164
14.7 Pickling 165
14.8 Pipes 166
14.9 Writing Modules 167
14.10 Debugging 168
14.11 Glossary 169
14.12 Exercises 169

15 Classes and Objects 172
15.1 User-Defined Types 172
15.2 Attributes 173
15.3 Rectangles 174
15.4 Instances as Return Values 176
15.5 Objects Are Mutable 176
15.6 Copying 177
15.7 Debugging 179
15.8 Glossary 179
15.9 Exercises 180

16 Classes and Functions 182
16.1 Time 182
16.2 Pure Functions 183
16.3 Modifiers 184
16.4 Prototyping versus Planning 185
16.5 Debugging 187
16.6 Glossary 188
16.7 Exercises 188

17 Classes and Methods 189
17.1 Object-Oriented Features 189
17.2 Printing Objects 190
17.3 Another Example 192
17.4 A More Complicated Example 192
Contents

17.5 The Init Method 193
17.6 The __str__ method 194
17.7 Operator Overloading 195
17.8 Type-Based Dispatch 195
17.9 Polymorphism 197
17.10 Debugging 198
17.11 Glossary 199
17.12 Exercises 199

18 Inheritance 201
18.1 Card Objects 201
18.2 Class Attributes 202
18.3 Comparing Cards 204
18.4 Decks 205
18.5 Printing the Deck 205
18.6 Add, Remove, Shuffle, and Sort 206
18.7 Inheritance 207
18.8 Class Diagrams 209
18.9 Debugging 210
18.10 Glossary 211
18.11 Exercises 212

19 Case Study: Tkinter 214
19.1 GUI 214
19.2 Buttons and Callbacks 215
19.3 Canvas Widgets 216
19.4 Coordinate Sequences 217
19.5 More Widgets 218
19.6 Packing Widgets 220
19.7 Menus and Callables 223
19.8 Binding 223
19.9 Debugging 226
19.10 Glossary 227
19.11 Exercises 228

Appendix 231
Index 241
Preface

THE STRANGE HISTORY OF THIS BOOK

In January 1999, I was preparing to teach an introductory programming class in Java. I had taught it three times and I was getting frustrated. The failure rate in the class was too high, and, even for students who succeeded, the overall level of achievement was too low.

One of the problems I saw was the books. I had tried three different books (and had read a dozen more), and they all had the same problems. They were too big, with too much unnecessary detail about Java and not enough high-level guidance about how to program. And they all suffered from the trap door effect: they would start out easy, proceed gradually, and then somewhere around Chapter 4 the bottom would fall out. The students would get too much new material, too fast, and I would spend the rest of the semester picking up the pieces.

Two weeks before the first day of classes, I decided to write my own book. I wrote one 10-page chapter a day for 13 days. I made some revisions on Day 14 and then sent it out to be photocopied.

My goals were:

- Keep it short. It is better for students to read 10 pages than not read 50 pages.
- Be careful with vocabulary. I tried to minimize the jargon and define each term at first use.
- Build gradually. To avoid trap doors, I took the most difficult topics and split them into a series of small steps.
- Focus on programming, not the programming language. I included the minimum useful subset of Java and left out the rest.

I needed a title, so on a whim I chose How to Think Like a Computer Scientist.
Preface

My first version was rough, but it worked. Students did the reading, and they understood enough that I could spend class time on the hard topics, the interesting topics, and (most important) letting the students practice.

I released the book under the GNU Free Documentation License, which allows users to copy, modify, and distribute the book.

What happened next is the cool part. Jeff Elkner, a high school teacher in Virginia, adopted my book and translated it into Python. He sent me a copy of his translation, and I had the unusual experience of learning Python by reading my own book.

Jeff and I revised the book, incorporated a case study by Chris Meyers, and in 2001 we released *How to Think Like a Computer Scientist: Learning with Python*, also under the GNU Free Documentation License. As Green Tea Press, I published the book and started selling hard copies through Amazon.com and college book stores. Other books from Green Tea Press are available at greenteapress.com.

In 2003, I started teaching at Olin College, and I got to teach Python for the first time. The contrast with Java was striking. Students struggled less, learned more, worked on more interesting projects, and generally had a lot more fun.

Over the last five years I have continued to develop the book, correcting errors, improving some of the examples, and adding material, especially exercises. In 2008, I started work on a major revision of the book – at the same time, I was contacted by an editor at Cambridge University Press who was interested in publishing the next edition. Good timing!

The result is this book, now with the less grandiose title *Python for Software Design*. Some of the changes are:

- I added a section about debugging at the end of each chapter. These sections present general techniques for finding and avoiding bugs, and warnings about Python pitfalls.
- I removed the material in the last few chapters about the implementation of lists and trees. I still love those topics, but I thought they were incongruent with the rest of the book.
- I added more exercises, ranging from short tests of understanding to a few substantial projects.
- I added a series of case studies – longer examples with exercises, solutions, and discussion. Some of them are based on Swampy, a suite of Python programs I wrote for use in my classes. Swampy, code examples, and some solutions are available from thinkpython.com.
- I expanded the discussion of program development plans and basic design patterns.
- The use of Python is more idiomatic. The book is still about programming, not Python, but now I think the book gets more leverage from the language.
I hope you enjoy working with this book, and that it helps you learn to program and think, at least a little bit, like a computer scientist.

ACKNOWLEDGMENTS

First and most importantly, I thank Jeff Elkner, who translated my Java book into Python, which got this project started and introduced me to what has turned out to be my favorite language.

I also thank Chris Meyers, who contributed several sections to *How to Think Like a Computer Scientist*.

And I thank the Free Software Foundation for developing the GNU Free Documentation License, which helped make my collaboration with Jeff and Chris possible.

I also thank the editors at Lulu who worked on *How to Think Like a Computer Scientist* and the editors at Cambridge University Press who worked on this edition.

I thank all the students who worked with earlier versions of this book and all the contributors (listed below) who sent in corrections and suggestions.

And I thank my wife, Lisa, for her work on this book, and Green Tea Press, and everything else, too.

CONTRIBUTOR LIST

More than 100 sharp-eyed and thoughtful readers have sent in suggestions and corrections over the past few years. Their contributions, and enthusiasm for this project, have been a huge help.

If you have a suggestion or correction, please send email to feedback@thinkpython.com. If I make a change based on your feedback, I will add you to the contributor list (unless you ask to be omitted).

If you include at least part of the sentence the error appears in, it will be easier for me to search for it. Page and section numbers are fine, too, but not quite as easy to work with. Thanks!

- Lloyd Hugh Allen sent in a correction to Section 8.4.
- Yvon Boulianne sent in a correction of a semantic error in Chapter 5.
- Fred Bremmer submitted a correction in Section 2.1.
- Jonah Cohen wrote the Perl scripts to convert the LaTeX source for this book into beautiful HTML.
- Michael Conlon sent in a grammar correction in Chapter 2 and an improvement in style in Chapter 1, and he initiated discussion on the technical aspects of interpreters.
- Benoit Girard sent in a correction to a humorous mistake in Section 5.6.
Courtney Gleason and Katherine Smith wrote horsebet.py, which was used as a case study in an earlier version of the book. Their program can now be found on the website.

Lee Harr submitted more corrections than we have room to list here, and indeed he should be listed as one of the principal editors of the text.

James Kaylin is a student using the text. He has submitted numerous corrections.

David Kershaw fixed the broken catTwice function in Section 3.10.

Eddie Lam has sent in numerous corrections to Chapters 1, 2, and 3. He also fixed the Makefile so that it creates an index the first time it is run and helped us set up a versioning scheme.

Man-Yong Lee sent in a correction to the example code in Section 2.4.

David Mayo pointed out that the word “unconsciously” in Chapter 1 needed to be changed to “subconsciously.”

Chris McAloon sent in several corrections to Sections 3.9 and 3.10.

Matthew J. Moelter has been a long-time contributor who sent in numerous corrections to and suggestions for the book.

Simon Dicon Montford reported a missing function definition and several typos in Chapter 3. He also found errors in the increment function in Chapter 13.

John Ouzts corrected the definition of “return value” in Chapter 3.

Kevin Parks sent in valuable comments and suggestions as to how to improve the distribution of the book.

David Pool sent in a typo in the glossary of Chapter 1, as well as kind words of encouragement.

Michael Schmitt sent in a correction to the chapter on files and exceptions.

Robin Shaw pointed out an error in Section 13.1, where the printTime function was used in an example without being defined.

Paul Sleigh found an error in Chapter 7 and a bug in Jonah Cohen’s Perl script that generates HTML from LaTeX.

Craig T. Snydal is testing the text in a course at Drew University. He has contributed several valuable suggestions and corrections.

Ian Thomas and his students are using the text in a programming course. They are the first ones to test the chapters in the latter half of the book, and they have made numerous corrections and suggestions.

Keith Verheyden sent in a correction in Chapter 3.

Peter Winstanley let us know about a longstanding error in our Latin in Chapter 3.

Chris Wrobel made corrections to the code in the chapter on file I/O and exceptions.

Moshe Zadka has made invaluable contributions to this project. In addition to writing the first draft of the chapter on dictionaries, he provided continual guidance in the early stages of the book.

Christoph Zwierschke sent several corrections and pedagogic suggestions and explained the difference between gleich and selbe.

James Mayer sent us a whole slew of spelling and typographical errors, including two in the contributor list.

Hayden McAfee caught a potentially confusing inconsistency between two examples.
Angel Arnal is part of an international team of translators working on the Spanish version of the text. He has also found several errors in the English version.

Tauhidul Hoque and Lex Berezhny created the illustrations in Chapter 1 and improved many of the other illustrations.

Dr. Michele Alzetta caught an error in Chapter 8 and sent some interesting pedagogic comments and suggestions about Fibonacci and Old Maid.

Andy Mitchell caught a typo in Chapter 1 and a broken example in Chapter 2.

Kalin Harvey suggested a clarification in Chapter 7 and caught some typos.

Christopher P. Smith caught several typos and is helping us prepare to update the book for Python 2.2.

David Hutchins caught a typo in the Preface.

Gregor Lingl is teaching Python at a high school in Vienna, Austria. He is working on a German translation of the book, and he caught a couple of bad errors in Chapter 5.

Julie Peters caught a typo in the Preface.

Florin Oprina sent in an improvement in makeTime, a correction in printTime, and a nice typo.

D. J. Webre suggested a clarification in Chapter 3.

Ken found a fistful of errors in Chapters 8, 9, and 11.

Ivo Wever caught a typo in Chapter 5 and suggested a clarification in Chapter 3.

Curtis Yanko suggested a clarification in Chapter 2.

Ben Logan sent in a number of typos and problems with translating the book into HTML.

Jason Armstrong saw a missing word in Chapter 2.

Louis Cordier noticed a spot in Chapter 16 where the code didn’t match the text.

Brian Cain suggested several clarifications in Chapters 2 and 3.

Rob Black sent in a passel of corrections, including some changes for Python 2.2.

Jean-Philippe Rey at Ecole Centrale Paris sent a number of patches, including some updates for Python 2.2 and other thoughtful improvements.

Jason Mader at George Washington University made a number of useful suggestions and corrections.

Jan Gundtofte-Bruun reminded us that “a error” is an error.

Abel David and Alexis Dinno reminded us that the plural of “matrix” is “matrices,” not “matrixes.” This error was in the book for years, but two readers with the same initials reported it on the same day. Weird.

Charles Thayer encouraged us to get rid of the semi-colons we had put at the ends of some statements and to clean up our use of “argument” and “parameter.”

Roger Sperberg pointed out a twisted piece of logic in Chapter 3.

Sam Bull pointed out a confusing paragraph in Chapter 2.

Andrew Cheung pointed out two instances of “use before def.”

C. Corey Capel spotted a missing word in the Third Theorem of Debugging and a typo in Chapter 4.

Alessandra helped clear up some Turtle confusion.

Wim Champagne found a brain-o in a dictionary example.

Douglas Wright pointed out a problem with floor division in arc.

Jared Spindor found some jetsam at the end of a sentence.

Lin Peiheng sent a number of very helpful suggestions.
Preface

Ray Hagtvedt sent in two errors and a not-quite-error.
Torsten Hübsch pointed out an inconsistency in Swampy.
Inga Petuhhov corrected an example in Chapter 14.
Arne Babenhausereheide sent several helpful corrections.
Mark E. Casida is is good at spotting repeated words.
Scott Tyler filled in a that was missing. And then sent in a heap of corrections.
Gordon Shephard sent in several corrections, all in separate emails.
Andrew Turner spotted an error in Chapter 8.
Adam Hobart fixed a problem with floor division in arc.
Daryl Hammond and Sarah Zimmerman pointed out that I served up math.pi too early. And Zim spotted a typo.
George Sass found a bug in a Debugging section.
Brian Bingham suggested Exercise 11.9.
Leah Engelbert-Fenton pointed out that I used tuple as a variable name, contrary to my own advice. And then found a bunch of typos and a “use before def.”
Joe Funke spotted a typo.
Chao-chao Chen found an inconsistency in the Fibonacci example.
Jeff Paine knows the difference between space and spam.
Lubos Pintes sent in a typo.
Gregg Lind and Abigail Heithoff suggested Exercise 14.6.
Max Hailperin has sent in a number of corrections and suggestions. Max is one of the authors of the extraordinary Concrete Abstractions, which you might want to read when you are done with this book.
Chotipat Pornavalai found an error in an error message.
Stanislaw Antol sent a list of very helpful suggestions.
Eric Pashman sent a number of corrections for Chapters 4–11.
Miguel Azevedo found some typos.
Jianhua Liu sent in a long list of corrections.
Nick King found a missing word.
Martin Zuther sent a long list of suggestions.
Adam Zimmerman found an inconsistency in my instance of an “instance” and several other errors.
Ratnakar Tiwari suggested a footnote explaining degenerate triangles.
Anurag Goel suggested another solution for is_abecedarian and sent some additional corrections. And he knows how to spell Jane Austen.
Kelli Kratzer spotted one of they typos.
Mark Griffiths pointed out a confusing example in Chapter 3.
Roydan Ongie found an error in my Newton’s method.
Patryk Wolowiec helped me with a problem in the HTML version.

Allen B. Downey
Needham, MA
Python for Software Design