

Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

For further volumes:

<http://www.springer.com/series/7592>

Kent D. Lee

Python Programming Fundamentals

 Springer

Dr. Kent D. Lee

Dept. Computer Science

Luther College

College Drive 700

52101 Decorah, IA

USA

kentdlee@luther.edu

Series editor

Ian Mackie

Advisory board

Samson Abramsky, University of Oxford, Oxford, UK

Chris Hankin, Imperial College London, London, UK

Dexter Kozen, Cornell University, Ithaca, USA

Andrew Pitts, University of Cambridge, Cambridge, UK

Hanne Riis Nielson, Technical University of Denmark, Lyngby, Denmark

Steven Skiena, Stony Brook University, Stony Brooks, USA

Iain Stewart, University of Durham, Durham, UK

ISSN 1863-7310

ISBN 978-1-84996-536-1

e-ISBN 978-1-84996-537-8

DOI 10.1007/978-1-84996-537-8

Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2010937477

© Springer-Verlag London Limited 2011

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licenses issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

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

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Computer Science is a creative, challenging, and rewarding discipline. Computer programmers, sometimes called software engineers, solve problems involving data: computing, moving, and handling large quantities of data are all tasks made easier or possible by computer programs. Money magazine ranked software engineer as the number one job in America in terms of flexibility, creativity, low stress levels, ease of entry, compensation, and job growth within the field [4].

Learning to program a computer is a skill that can bring you great enjoyment because of the creativity involved in designing and implementing a solution to a problem. Python is a good first language to learn because there is very little overhead in learning to write simple programs. Python also has many libraries available that make it easy to write some very interesting programs including programs in the areas of Computer Graphics and Graphical User Interfaces: two topics that are covered in this text.

In this text, students are taught to program by giving them many examples and practice exercises with solutions that they can work on in an interactive classroom environment. The interaction can be accomplished using a computer or using pen and paper. By making the classroom experience active, students reflect on and apply what they have read and heard in the classroom. By using a skill or concept right away, students quickly discover if they need more reinforcement of the concept, while teachers also get immediate feedback. There is a big difference between seeing a concept demonstrated and using it yourself and this text encourages applying concepts immediately to test understanding. This is vital in Computer Science since new skills and concepts build on what we have already learned.

In several places within this book there are examples presented that highlight patterns of programming. These patterns appear over and over in programs we write. In this text, patterns like the *Accumulator Pattern* and the *Guess and Check Pattern* are presented and exercises reinforce the recognition and application of these and other abstract patterns used in problem-solving. Learning a language is certainly one important goal of an introductory text, but acquiring the necessary problem-solving skills is even more important. Students learn to solve problems on their own by recognizing when certain patterns are relevant and then applying these patterns in their own programs.

Recent studies in Computer Science Education indicate the use of a debugger can greatly enhance a student's understanding of programming [1]. A debugger is a tool that

lets the programmer inspect the state of a program at any point while it is executing. There is something about actually seeing what is happening as a program is executed that helps make an abstract concept more concrete. This text introduces students to the use of a debugger and includes exercises and examples that show students how to use a debugger to discover how programs work.

There are additional resources available for instructors teaching from this text. They include lecture slides and a sample schedule of lectures for a semester long course. Solutions to all programming exercises are also available upon request. Visit <http://cs.luther.edu/~leekent/CS1> for more information.

Python is a good language for teaching introductory Computer Science because it is very accessible and can be incrementally taught so students can start to write programs before having to learn the whole language. However, at the same time, Python is also a developing language. Python 3.1 was recently released to the public. This release of Python included many performance enhancements which were very good additions to the language. There were also some language issues with version 2.6 and earlier that were cleaned up at the same time that were not backwards compatible. The result is that not all Python 2 programs are compatible with Python 3 and vice versa. Because both Python 2 and Python 3 are in use today, this text will point out the differences between the two versions where appropriate. These differences will be described by inset boxes titled `Python 2 ~> 3` within the text where the differences are first encountered.

It is recommended that students reading this text use Python 3.1 or later for writing and running their programs. All Python programs presented in the text are Python 3 programs. The libraries used in this text all work with Python 3. However, there may be some libraries that have not been ported to Python 3 that a particular instructor would like to use. In terms of what is covered in this text, the differences between Python 2 and 3 are pretty minor and either language implementation will work to use with the text.

Acknowledgments

I'd like to thank David Ranum, who encouraged me to write a text on introductory programming that included some of the projects I've developed, and Brad Miller, who suggested we try Python as an introductory language. It's wonderful to work with colleagues that care as much as I do about teaching students a rewarding and creative discipline. But most importantly, I would like to thank the students who have taken classes from me. Teaching has given me back much more than the many hours I have invested in it and seeing a student grow in his or her programming skills is its own reward.

I would especially like to thank Nathaniel Lee, who not only let his dad teach him, but was a great sounding board and test subject for this text. Thank you, Nathan, for all your valuable feedback and for your willingness to learn.

Credits

At times in this text Microsoft Windows is referred to when installing software. Windows is a registered trademark of Microsoft Corporation in the United States and other countries. Mac OS X is referred to at times within this text. Mac and Mac OS are trademarks of Apple Inc., registered in the U.S. and other countries. The Garmin Forerunner and Garmin Training Center are referred to in Chap. 4. Garmin is a registered trademark, and Forerunner and Training Center are trademarks of Garmin Ltd. or its subsidiaries in the United States and other countries.

This book also introduces readers to Wing IDE 101, which is used in examples throughout the text. Wing IDE 101 is a free simplified edition of Wing IDE Professional, a full-featured integrated development environment designed specifically for Python. For more information on Wing IDE, see www.wingware.com. Wingware and Wing IDE are trademarks or registered trademarks of Wingware in the United States and other countries.

Suggestions

I welcome suggestions for future printings of this text. If you like this text and have suggestions for future printings, please write up your suggestion(s) and email them to me. The more complete your write up, the more likely I will be to consider your suggestion. If I select your suggestion for a future printing I'll be sure to include your name in the preface as a contributor to the text. Suggestions can be emailed to kentdlee@luther.edu or kentdlee@gmail.com.

Contents

1	Introduction	1
1.1	The Python Programming Language	2
1.2	Installing Python and Wing IDE 101	3
1.3	Writing Your First Program	7
1.4	What is a Computer?	8
1.5	Binary Number Representation	10
1.6	What is a Programming Language?	13
1.7	Hexadecimal and Octal Representation	15
1.8	Writing Your Second Program	16
1.9	Syntax Errors	18
1.10	Types of Values	19
1.11	The Reference Type and Assignment Statements	20
1.12	Integers and Real Numbers	21
1.13	Strings	24
1.14	Integer to String Conversion and Back Again	25
1.15	Getting Input	26
1.16	Formatting Output	27
1.17	When Things Go Wrong	30
1.18	Review Questions	33
1.19	Exercises	34
1.20	Solutions to Practice Problems	36
2	Decision Making	41
2.1	Finding the Max of Three Integers	45
2.2	The Guess and Check Pattern	47
2.3	Choosing from a List of Alternatives	48
2.4	The Boolean Type	50
2.5	Short Circuit Logic	53
2.6	Comparing Floats for Equality	54
2.7	Exception Handling	55
2.8	Review Questions	57

2.9	Exercises	58
2.10	Solutions to Practice Problems	61
3	Repetitive Tasks	65
3.1	Operators	67
3.2	Iterating Over a Sequence	69
3.3	Lists	71
3.4	The Guess and Check Pattern for Lists	74
3.5	Mutability of Lists	75
3.6	The Accumulator Pattern	79
3.7	Reading from and Writing to a File	80
3.8	Reading Records from a File	82
3.9	Review Questions	85
3.10	Exercises	86
3.11	Solutions to Practice Problems	88
4	Using Objects	93
4.1	Constructors	96
4.2	Accessor Methods	98
4.3	Mutator Methods	98
4.4	Immutable Classes	99
4.5	Object-Oriented Programming	100
4.6	Working with XML Files	100
4.7	Extracting Elements from an XML File	103
4.8	Dictionaries	104
4.9	Getting at the Data in an XML File	105
4.10	Working with Time	107
4.11	Parallel Lists	107
4.12	Visualizing a Workout	108
4.13	Review Questions	110
4.14	Exercises	110
4.15	Solutions to Practice Problems	113
5	Defining Functions	119
5.1	Why Write Functions?	120
5.2	Passing Arguments and Returning a Value	121
5.3	Scope of Variables	122
5.4	The Run-time Stack	126
5.5	Mutable Data and Functions	129
5.6	Predicate Functions	131
5.7	Top-Down Design	133
5.8	Bottom-Up Design	134
5.9	Recursive Functions	134
5.10	The Main Function	136
5.11	Keyword Arguments	139

5.12	Default Values	140
5.13	Functions with Variable Number of Parameters	140
5.14	Dictionary Parameter Passing	141
5.15	Review Questions	142
5.16	Exercises	142
5.17	Solutions to Practice Problems	145
6	Event-Driven Programming	149
6.1	The Root Window	150
6.2	Menus	151
6.3	Frames	152
6.4	The Text Widget	152
6.5	The Button Widget	153
6.6	Creating a Reminder!	154
6.7	Finishing up the Reminder! Application	156
6.8	Label and Entry Widgets	157
6.9	Layout Management	158
6.10	Message Boxes	159
6.11	Review Questions	160
6.12	Exercises	161
6.13	Solutions to Practice Problems	164
7	Defining Classes	167
7.1	Creating an Object	168
7.2	Inheritance	172
7.3	A Bouncing Ball Example	178
7.4	Polymorphism	180
7.5	Getting Hooked on Python	181
7.6	Review Questions	184
7.7	Exercises	184
7.8	Solutions to Practice Problems	190
	Appendix A Integer Operators	193
	Appendix B Float Operators	195
	Appendix C String Operators and Methods	197
	Appendix D List Operators and Methods	201
	Appendix E Dictionary Operators and Methods	203
	Appendix F Turtle Methods	205
	Appendix G TurtleScreen Methods	217
	Appendix H The Reminder! Program	225

Appendix I The Bouncing Ball Program	227
Glossary	231
References	237
Index	239