

INTRODUCTION TO COMPUTING SCIENCE AND PROGRAMMING I

FALL 2010 EDITION

BY
GREG BAKER

Faculty of Applied Sciences
Simon Fraser University
© Greg Baker, 2004-2010

Copyright © 2004–2010 Greg Baker.

This work is licenced under the Creative Commons Attribution-ShareAlike 2.5 Canada License. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-sa/2.5/ca/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

CONTENTS

COURSE INTRODUCTION	11
Learning Resources	12
Requirements	13
Evaluation	14
About the Author	15
I Computer Science and Programming	17
1 COMPUTING SCIENCE BASICS	19
1.1 What is an Algorithm?	19
1.2 What is Computing Science?	22
1.3 What is Programming?	25
1.4 Pseudocode	26
Summary	28
2 PROGRAMMING BASICS	29
2.1 Starting with Python	29
2.2 Doing Calculations	32
2.3 Storing Information	35
2.4 Types	36
2.5 User Input	40
2.6 How Computers Represent Information	41
2.7 Example Problem Solving: Feet and Inches	48
Summary	53

3	CONTROL STRUCTURES	55
3.1	Making Decisions	55
3.2	Definite Iteration: <code>for</code> loops	59
3.3	Indefinite Iteration: <code>while</code> loops	62
3.4	Choosing Control Structures	63
3.5	Example Problem Solving: Guessing Game	65
3.6	Running Time	70
3.7	Debugging	77
3.8	Coding Style	80
3.9	More About Algorithms	84
	Summary	86
4	FUNCTIONS AND DECOMPOSITION	89
4.1	Defining Functions	89
4.2	Variable Scope	94
4.3	Python Modules	97
4.4	Objects	98
4.5	Handling Errors	102
	Summary	103
II	Problem Solving	105
5	DATA STRUCTURES	107
5.1	Lists	107
5.2	Lists and <code>for</code> loops	111
5.3	Slicing and Dicing	112
5.4	Strings	114
5.5	Mutability	115
5.6	References	117
	Summary	122

6	ALGORITHMS	123
6.1	Searching	123
6.2	Sorting	126
6.3	Recursion	131
6.4	Designing with Recursion	135
6.5	What isn't computable?	142
	Summary	144
7	WORKING WITH FILES	147
7.1	File Output	147
7.2	File Input	150
7.3	The Operating System	154
7.4	Disks and Files	155
7.5	Example Problem Solving: File Statistics	158
	Summary	164
III	Appendices	165
A	TECHNICAL INSTRUCTIONS	167
A.1	Installing Python	167
A.2	Using Python	168
A.3	Computing Accounts	171
A.4	Working in the Lab	172
	Summary	173

LIST OF FIGURES

1.1	The “creaming method”: an everyday algorithm.	20
1.2	An algorithm that guesses a secret number between 1 and 100.	24
1.3	Figure 1.2 written in pseudocode.	26
1.4	Another pseudocode example: a digital clock	27
2.1	Prefixes for storage units.	42
2.2	The four-bit unsigned integer values.	44
2.3	Some examples of binary addition	44
2.4	The four-bit two’s complement values	45
2.5	Conversion of the string “Hi” to binary.	47
2.6	Meters to feet-inches conversion pseudocode.	49
2.7	Converting to feet and inches: number of feet.	49
2.8	Converting to feet and inches: feet and inches.	50
2.9	Converting to feet and inches: rounding inches.	51
2.10	Converting to feet and inches: printing quotes	52
2.11	Converting to feet and inches: fixed rounding error	53
3.1	Simplified guessing game	56
3.2	Implementation of Figure 3.1	57
3.3	Pseudocode to calculate factorials	60
3.4	Using a <code>for</code> loop in Python	60
3.5	Calculating factorials with Python	61
3.6	Using a <code>while</code> loop in Python	62
3.7	Guessing game: first guess	65
3.8	Guessing game: get an answer	66
3.9	Guessing game: trying a loop	67
3.10	Guessing game: a working loop	68
3.11	Guessing game: final version	69

3.12	A much worse version of the guessing game	70
3.13	Algorithm to check for repeated letters in a word	73
3.14	Repeated letters Python implementation	73
3.15	Algorithm to solve the subset-sum problem.	75
3.16	Running time of an exponential algorithm	75
3.17	Graph of the functions $\log_2 n$, n , and n^2	77
3.18	A program with poor style	82
3.19	Figure 3.18 with the style improved	83
3.20	The number 13 represented with 8 bits	85
3.21	Pseudocode to determine the 8-bit binary representation	86
4.1	A program with a function defined	90
4.2	A sample function	92
4.3	A program that takes advantage of local variables	95
4.4	Program that prints today's date.	97
4.5	Date manipulation with the <code>datetime</code> module's objects	100
4.6	Catching an exception	102
4.7	Asking until we get correct input	103
5.1	Building a list with the <code>append</code> method	111
5.2	Iterating over a list	112
5.3	Manipulating strings and lists	116
5.4	Variables referencing their contents	118
5.5	Reference copied during assignment: aliasing	119
5.6	Changing either alias changes both	119
5.7	An expression creates a new reference and breaks the alias	120
5.8	A calculation creates a new instance containing the results	121
5.9	Slicing a list forces copying of its elements	121
6.1	Python implementation of linear search	124
6.2	Python implementation of binary search	127
6.3	Checking for repeated letters with sorting	128
6.4	Graph of the functions n^2 and $n \log_2 n$	129
6.5	Algorithm for selection sort	131
6.6	Python implementation of selection sort	132
6.7	Recursively calculate factorials	132
6.8	Functions calls made to calculate <code>factorial(3)</code>	133
6.9	Pseudocode for a recursive algorithm	138

6.10	Recursively reversing a string	139
6.11	Recursively putting spaces between characters in a string . . .	141
6.12	Fooling a function that claims to solve the halting problem. . .	143
7.1	Writing text to a file	148
7.2	File created by Figure 7.1	148
7.3	Writing text to a file	149
7.4	File created by Figure 7.3	150
7.5	Reading a text file and counting line lengths	151
7.6	Program to read a series of times in a file	153
7.7	Input file (<code>times.txt</code>) for Figure 7.6	153
7.8	Output of Figure 7.6	153
7.9	Communication between the user, applications, OS, and hard- ware	155
7.10	A disk divided into blocks	157
7.11	The blocks of a file defragmented	158
7.12	Word count: open file and count lines	159
7.13	Word count test file	159
7.14	Word count: counting characters 1	160
7.15	Word count: counting characters 2	161
7.16	Word count: final program	163
A.1	IDLE, after running a “Hello World” statement	169
A.2	IDLE, displaying a syntax error	170
A.3	IDLE, displaying a run-time error	171

COURSE INTRODUCTION

[Most of the material in this introduction will apply to all offerings of CMPT 120, but some details will vary depending on the semester and instructor. Please check your course web site for details.]

Welcome to CMPT 120, “Introduction to Computing Science and Programming I”. This course is an introduction to the core ideas of computing science and the basics of programming. This course is intended for students who do not have programming experience. If you have done a significant amount of programming, you should take CMPT 126 instead.

This course isn’t intended to teach you how to use your computer. You are expected to know how to use your computer: you should be comfortable using it for simple tasks such as running programs, finding and opening files, and so forth. You should also be able to use the Internet.

Here are some of the goals set out for students in this course. By the end of the course, you should be able to:

- Explain some of the basic ideas of computing science.
- Explain what computer programming is.
- Create algorithms to solve simple problems.
- Implement computer programs in the Python programming language.
- Apply the core features of a programming language to solve problems.
- Design programs that are easy to understand and maintain.

Keep these goals in mind as you progress through the course.

LEARNING RESOURCES

STUDY GUIDE

The *Study Guide* is intended to guide you through this course. It will help you learn the content and determine what information to focus on in the texts.

Some suggested readings for each section are listed at the beginning of the units. The instructor for your section of the course may modify these readings. You should also look at the key terms listed at the end of each unit.

In some places, there are references to other sections. A reference to “Topic 3.4,” for example, means Topic 4 in Unit 3.



In the *Study Guide*, side-notes are indicated with this symbol. They are meant to replace the asides that usually happen in lectures when students ask questions. They aren’t strictly part of the “course.”

REQUIRED TEXTS

The only required text for this course (besides this *Study Guide*) is *How to Think Like a Computer Scientist: Learning With Python*. It is available from the SFU bookstore, or you can download it from the course web site. This book is a nice introduction to Python programming, which we will use in the second half of the course.

The title of the book is a little misleading. The book does *not* discuss computer science; it only covers computer programming. In this Guide, we will try to cover more actual “computing science.”

RECOMMENDED TEXTS

There are currently no recommended texts for this course. If additional resources are required during the semester, the instructor may recommend other books and place them on reserve at the library.

WEB MATERIALS

The web materials for this course can be found in Computing Science's "Course Central," <http://www.cs.sfu.ca/CC/>.

ONLINE REFERENCES

There are several online references that are as important as the texts. You can find links to them on the course web site.

These resources are very important to your success in this course. They aren't meant to be read from beginning to end like the readings in the textbook. You should use them to get an overall picture of the topic and as references as you do the assignments.

EMAIL COMMUNICATIONS

The TAs and course instructor will use email to send announcements and tips during the semester. You should read your SFU email account regularly, at least a few times each week.

You can also contact the TAs and course instructor by email if you need help. See the "Getting Help" section at the end of the Introduction for details.

REQUIREMENTS

COMPUTER REQUIREMENTS

You need to have access to a computer for this course. The labs on campus or your own computer can be used. Instructions on using the labs can be found on the course web site.

You will also need a connection to the Internet through a dial-up connection (one is provided with your SFU registration) or through another type of connection like a cable modem or a DSL line. A high-speed connection will make your life easier, but it isn't required.

You should activate your SFU computing account as soon as possible if you haven't done so already. Instructions can be found in your registration material.

SOFTWARE REQUIREMENTS

All of the software that you need for this course can be downloaded for free. Links to obtain the course software are on the course web site. Appendix A contains instructions for installing and working with the Python software.

There may be some other pieces of software that you'll need to use when submitting your work. See the course web site for instructions on installing and using these.

EVALUATION

MARKING

The marking scheme for the course will be announced in class or by email in the first week.

LABS AND ASSIGNMENTS

You will have weekly labs in this course. Details will be announced in class. The labs will consist of relatively short exercises that you should be able to complete quickly. They are intended to make sure you stay up to date with the material in the course as it progresses.

The assignments are more involved than the labs. You will have to figure out more on your own and explore the concepts on the course.

EXAMS

There will be two examinations in this course. They are closed-book, i.e. you aren't allowed any notes, calculators, or other aids. The exams have a mixture of short and longer answer questions.

The 50 minute midterm exam will be held in week 7 or 8. The instructor will announce what material will be covered in the midterm a few weeks before. The final exam will be three hours long and will cover all of the course material.

ACADEMIC DISHONESTY

We take academic dishonesty very seriously in the School of Computing Science. Academic dishonesty includes (but is not limited to) the following: copying another student's assignment; allowing others to complete work for you; allowing others to use your work; copying part of an assignment from an outside source; cheating in any way on a test or examination.

If you are unclear on what academic dishonesty is, please read Policy 10.02. It can be found in the "Policies & Procedures" section in the SFU web site.

Cheating on a lab or assignment will result in a mark of 0 on the piece of work. At the instructor's option, further penalties may be requested. Any academic dishonesty will also be recorded in your file, as is required by University policy.

Any academic dishonesty on the midterm or final will result in a recommendation that an F be given for the course.

ABOUT THE AUTHOR

The author hates writing about himself, so he isn't going to. So there.

I'd like to thank Arsalan Butt, Yaroslav Litus and Mashaal Mamemon who worked on the creation of this course with me in the summer of 2004. Thanks also to Toby Donaldson, Anne Lavergne, David Mitchell, Brad Bart, John Edgar, and all of the other faculty members around the School that were willing to listen to an idea and, more often than not, reply with a better one.

If you have any comments on this guide, please feel free to contact me at ggbaker@cs.sfu.ca.

