Introduction to Computing and Programming in Python: A Multimedia Approach

Mark Guzdial

College of Computing/GVU
Georgia Institute of Technology

PRENTICE HALL, *Upper Saddle River, New Jersey 07458*

Dedicated to my wife,
Barbara Jane Ericson.

Contents

APPENDICES

List of Figures

0.1 TO TEACHERS
0.2 TYPOGRAPHICAL NOTATIONS
0.3 ACKNOWLEDGEMENTS

Preface

One of the clearest lessons from the research on computing education is that one doesn't just "learn to program." One learns to program *something* [5, 18], and the motivation to do that something can make the difference between learning to program or not [7]. People want to communicate. We are social creatures, and the desire to communicate is one of our primal motivations. Increasingly, the computer is used as a tool for communication, even more than a tool for calculation. Virtually all published text, images, sounds, music, and movies today are prepared using computing technology.

This book is about teaching people to program in order to communicate. The book focuses how to manipulate images, sounds, text, and movies as professionals might, but with programs written by the students. There are no illusions here: most people will use professional-grade applications to perform these same manipulations. But knowing *how* to do it with your own programs means that you *can* do it if you need to. Want to say something with your media, but you don't know how to make PhotoShop or Final Cut Pro do what you want? Knowing how to program means that you have power of expression that is not limited by the application software.

It might also be true that knowing how the algorithms in the media applications work allows you to use them better, or move from one application to the next more easily. If your focus in an application is what menu item does what, every application is different. But if your focus is to move or color the pixels in the way that you want, then maybe it's easier to get past the menu items and focus on what you want to say.

This book is not just about programming in media. Media manipulation programs can get hard to write, or behave in unexpected ways. Natural questions arise like "Why is this same image filter faster in Photoshop?" and "That was hard to debug–are there ways of writing programs that are *easier* to debug?" Answering questions like these is what computer scientists do. There are several chapters at the end of the book that are about *computing*, not just programming, and more generally than just media.

The computer is the most amazingly creative device that humans have ever conceived of. It is literally completely made up of mind-stuff. The notion "Don't just dream it, be it" is really possible on a computer. If you can imagine it, you can make it "real" on the computer. Playing with programming can be and *should* be enormous fun.

0.1 TO TEACHERS

The curricular content of this book matches the "Imperative-first" approach described in the ACM/IEEE *Computing Curriculum 2001* standards document [4]. The book starts with a focus on fundamental programming constructs: assignments, sequential operations, iteration, conditionals, and defining functions. Abstractions (such as algorithmic complexity, program efficiency, computer organization, hierarchical decomposition, recursion, and object-oriented programming) are emphasized more later, after the students have a context for understanding them.

The reason for this unusual ordering is research in learning sciences. Memory is associative–we remember things based on what else we relate to those things. People can learn concepts and skills on the promise that it will be useful some day, but those concepts and skills will be related only to those promises, not to everyday life. The result has been described as "brittle knowledge" [8]–the kind of knowledge that gets you through the exam, but promptly gets forgotten because it doesn't relate to anything but being in that class. If we want students to gain *transferable* knowledge (knowledge that can be applied in new situations), we have to help them to relate the knowledge to more general problems, so that the memories get indexed in ways that associate with those kinds of problems [22]. Thus, we teach with concrete experiences that students can explore and relate to (e.g., iteration for removing red-eye in pictures), and later lay abstractions on top of that (e.g., achieving the same goal using recursion or functional filters and maps). But even the concrete experiences are first anchored in relevant contexts.

We do know that starting from the abstractions doesn't really work for students. Ann Fleury has shown that novice students just don't buy what we tell them about encapsulation and reuse (e.g., [11]). Students prefer simpler code that they can trace easily, and actually think that such code is *better*. It takes time and experience for students to realize that there is value in well-designed systems, and without experience, it's very difficult for students to learn the abstractions.

The *media computation* approach used in this book is to start from what people use computers for: Image manipulation, exploring digital music, viewing and creating web pages, and so on. We then explain programming and computing in terms of these activities. We want students to visit Amazon (for example) and think, "Here's a catalog website–and I know that these are implemented with a database and a set of programs that format the database entries as Web pages." Starting from a relevant context makes transfer of knowledge and skills more likely, but it also helps with retention.

The media computation approach spend about 2/3 of the time on giving students experiences with a variety of media in contexts that they find motivating. After that 2/3, though, they start to develop questions. "Why is that Photoshop is faster than my program?" and "Movie code is slow – how slow do programs get?" are typical. At that point, we introduce the abstractions and the valuable insights from Computer Science that answer *their* questions. That's what the last part of this book is about.

A different body of research in computing education has been exploring why withdrawal or failure rates in introductory computing have been so high. One of the common themes is that computing courses seem "irrelevant" and unnecessarily focusing on "tedious details" such as efficiency [26][1]. A communications context is perceived as relevant by the students (as they tell us in surveys and interviews [13][23]). The relevant context is part of the explanation for the success we have had with retention in the Georgia Tech course for which this book was written.

The abstraction-late ordering isn't the only unusual ordering in this approach. We start using arrays and matrices in chapter 3, in our first significant programs. Typically, introductory computing courses push arrays off until later, since they're obviously more complicated than variables with simple values. But a relevant context is very powerful [18]. The matrices of pixels in images occur in the students' everyday life–a magnifying class on a computer monitor or television makes that clear.

The rate of students withdrawing from introductory computing courses or receiving a D or F grade (commonly called the *WDF rate*) has been reported in the 30–50% range, or even higher. At Georgia Tech, from 2000–2002, we had an average WDF rate of 28% in our introductory course which was required by all majors. We use this text in our course *Introduction to Media Computation.* Our first pilot offering of the course had 121 students, no computing or engineering majors, and 2/3 of the course was female. Our WDF rate was 11.5%. Spring 2004 was the first semester taught by instructors other than the author, and the WDF rate dropped to 9.5% for the 395 students who enrolled. Charles Fowler at Gainesville College in Georgia has been having similar results in his courses there.

Our publisher, Alan Apt of Prentice-Hall, recognizes that this book represents a new and radical approach to teaching introductory computing. **The publisher is willing to provide textbooks at no cost for a trial offering of a course (or a section of a large course)** to encourage you to try this approach in your own school.

0.1.1 Ways to Use This Book

This book represents what we teach at Georgia Tech in pretty much the ordering that we use. Individual teachers may skip some sections (e.g., the section on additive synthesis, MIDI, and MP3), but all of the content here has been tested with our students.

However, we can imagine using this material in many other ways:

- A short introduction to computing could be taught with just chapters 2 (introduction to programming) and 3 (introduction to image processing), perhaps with some material from chapters 4 and 5. We have taught even single day workshops on media computation using just this material.

- Chapters 6 through 8 basically replicate the computer science concepts from chapters 3 through 5, but in the context of sounds rather than images. We find the replication useful–some students seem to relate better to the concepts of iteration and conditionals better when working with one medium than the other. Further, it gives us the opportunity to point out that the same *algorithm* can have similar effects in different media (e.g., scaling a picture up or down and shifting a sound higher or lower in pitch is the same algorithm). But it could certainly be skipped to save time.

- Chapter 12 (on movies) introduces no new programming or computing concepts. While motivating, movie processing could be skipped for time.

- We do recommend getting to at least some of the chapters in the last unit, in order to lead students into thinking about the computing and programming in a more abstract manner, but clearly not *all* of the chapters have to be covered.

0.1.2 Python and Jython

The programming language used in this book is Python. Python has been described as "executable pseudo-code." We have found that Python is learnable and usable by non-CS majors (and presumably, by Computer Science majors as well), and since it's actually used for communications tasks (e.g., Web site development), it's a relevant language for an introductory computing course. For example, job advertisements posted to the Python website (`http://www.python.org`) show that companies like Google and Industrial Light & Magic hire Python programmers.

The specific dialect of Python used in this book is *Jython* (`http://www.jython.org`). Jython *IS* Python. The differences between Python (normally implemented in C) and Jython (which is implemented in Java) are akin to the differences between any two language implementations (e.g., Microsoft vs. GNU C++ implementations)–the basic language is *exactly* the same, with some library and details differences that most students will never notice.

0.2 TYPOGRAPHICAL NOTATIONS

Examples of Python code look like this: `x = x + 1`. Longer examples look look like this:

```
def helloWorld():
  print "Hello, world!"
```

When showing something that the user types in with Python's response, it will have a similar font and style, but the user's typing will appear after a Python prompt (`>>>`):

```
>>> print 3 + 4
7
```

User interface components of JES (Jython Environment for Students) will be specified using a smallcaps font, like SAVE menu item and the LOAD button.

There are several special kinds of sidebars that you'll find in the book.

Recipe 1: An Example Recipe

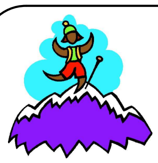
Recipes (programs) appear like this:

```
def helloWorld():
  print "Hello, world!"
```

Computer Science Idea: An Example Idea
Key computer science concepts appear like this.

Common Bug: An Example Common Bug
Common things that can cause your recipe to fail appear like this.

Debugging Tip: An Example Debugging Tip
If there's a good way to keep those bugs from creeping into your recipes in the first place, they're highlighted here.



Making it Work Tip: An Example How To Make It Work
Best practices or techniques that really help are highlighted like this.

0.3 ACKNOWLEDGEMENTS

Our sincere thanks go out to the following:

- Jason Ergle, Claire Bailey, David Raines, and Joshua Sklare who made JES a reality with amazing quality in an amazingly short amount of time. Jason and David took JES the next steps, improving installation, debugging, and process support. Adam Wilson and Toby Ho added the wonderful support for identifying blocks, MIDI music, and a debugger. Eric Mickley improved the error messages significantly. Keith McDermott gave us MovieMaker and worked on the picture support. Adam has been the caretaker of the project and brought it to the point it is today.

- Adam Wilson built the MediaTools that are so useful for exploring sounds and images and processing video.

- Andrea Forte, Mark Richman, Matt Wallace, Alisa Bandlow, Derek Chambless, Larry Olson, and David Rennie helped build course materials. Derek, Mark, and Matt created many example programs. Barbara Ericson reviewed the book as we worked on the Java version together, and made numerous suggestions and improvements.

- There were several people who really made the effort come together at Georgia Tech. Bob McMath, Vice-Provost at Georgia Tech, and Jim Foley, Associate Dean for Education in the College of Computing, invested in this effort early on. Kurt Eiselt worked hard to make this effort real, convincing others to take it seriously. Janet Kolodner and Aaron Bobick were excited and encouraging about the idea of media computation for students new to computer science. Jeff Pierce reviewed and advised us on the design of the media functions used in the book. Aaron Lanterman gave me lots of advice on how to convey the digital material content accurately.

- Joan Morton, Chrissy Hendricks, and all the staff of the GVU Center made sure that we had what we needed and that the details were handled to make this effort come together.
- Charles Fowler was the first person outside of Georgia Tech willing to take the gamble and trial the course in his own institution (Gainesville College), for which we're very grateful.
- The pilot course offered in Spring 2003 at Georgia Tech was very important in helping us improve the course. Andrea Forte, Rachel Fithian, and Lauren Rich did the assessment of the pilot offering of the course, which was incredibly valuable in helping us understand what worked and what didn't. The first Teaching Assistants (Jim Gruen, Angela Liang, Larry Olson, Matt Wallace, Adam Wilson, and Jose Zagal) did a lot to help create this approach. Blair MacIntyre, Colin Potts, and Monica Sweat helped make the materials easier for others to adopt.
- Many students pointed out errors and made suggestions to improve the book. Thanks to Catherine Billiris, Jennifer Blake, Karin Bowman, Maryam Doroudi, Suzannah Gill, Baillie Homire, Jonathan Laing, Mireille Murad, Michael Shaw, Summar Shoaib, and especially Jonathan Longhitano who has a real flair for oopy-editing.
- Most of the clip art is used with permission from the *Art Explosion* package by Nova Development.
- Thanks for permission to use their snapshots from class in examples are former *Media Computation* students Constantino Kombosch, Joseph Clark, and Shannon Joiner.
- Finally but most importantly, Barbara Ericson, and Matthew, Katherine, and Jennifer Guzdial, who allowed themselves to be photographed and recorded for Daddy's media project and were supportive and excited about the class.