
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*



Copyright held by Mark Guzdial, 2003 & 2004.





Dedicated to my wife,
Barbara Jane Ericson.



Contents

Contents	iv
Preface	1
0.1 To Teachers	1
0.1.1 Ways to Use This Book	3
0.1.2 Python and Jython	4
0.2 Typographical notations	4
0.3 Acknowledgements	5
I Introduction	7
1 Introduction to Computer Science and Media Computation	8
1.1 What is computer science about?	8
1.2 What Computers Understand	12
1.3 Media Computation: Why digitize media?	15
1.4 Computer Science for Everyone	16
1.4.1 It's about communication	16
1.4.2 It's about process	17
2 Introduction to Programming	19
2.1 Programming is about Naming	19
2.1.1 Files and their Names	20
2.2 Programming in Python	21
2.3 Programming in JES	22
2.4 Media Computation in JES	24
2.4.1 Showing a Picture	27
2.4.2 Playing a Sound	28
2.4.3 Naming your Media (and other Values)	29
2.5 Making a Recipe	33
2.5.1 Variable Recipes: Real functions that Take Input	37
II Pictures	41
3 Modifying Pictures using Loops	42
3.1 How Pictures are Encoded	42
3.2 Manipulating Pictures	47
3.2.1 Exploring pictures	52
3.3 Changing color values	55
3.3.1 Using loops in pictures	55
3.3.2 Increasing/decreasing red (green, blue)	58
3.3.3 Creating a sunset	64



	v
3.3.4 Making sense of functions	65
3.3.5 Lightening and darkening	69
3.3.6 Creating a negative	70
3.3.7 Converting to grayscale	71
4 Modifying pixels in a range	74
4.1 Copying pixels	74
4.1.1 Looping across the pixels with range	75
4.1.2 Mirroring a picture	76
4.2 Copying and transforming pictures	81
4.2.1 Copying	82
4.2.2 Creating a Collage	88
4.2.3 Blending Pictures	92
4.2.4 Rotation	94
4.2.5 Scaling	96
4.3 Replacing Colors	100
4.3.1 Reducing red eye	101
4.3.2 Sepia toned and posterized pictures: Using conditionals to choose the color	102
4.4 Combining pixels: Blurring	107
4.5 Color Figures	112
5 Combining Pictures	123
5.1 Background subtraction	123
5.2 Chromakey	124
5.3 Drawing on images with pixels	127
5.4 Drawing with drawing commands	129
5.4.1 Vector and Bitmap Representations	130
5.5 Programs as Specifying Drawing Process	131
5.5.1 Why do write programs?	134
III Sounds	137
6 Modifying Sounds using Loops	138
6.1 How Sound is Encoded	138
6.1.1 The Physics of Sound	138
6.1.2 Exploring how sounds look	142
6.1.3 Encoding the Sound	144
6.2 Manipulating sounds	149
6.2.1 Open sounds and manipulating samples	149
6.2.2 Using MediaTools for looking at captured sounds	152
6.2.3 Introducing the loop	155
6.3 Changing the volume of sounds	158
6.3.1 Increasing volume	158
6.3.2 Did that really work?	159
6.3.3 Decreasing volume	164



6.3.4	Making sense of functions, in sounds	164
6.3.5	Normalizing sounds	165
6.3.6	Generating clipping	167
7	Modifying Samples in a Range	168
7.0.7	Manipulating different sections of the sound differently	168
7.0.8	Splicing sounds	170
7.0.9	Backwards sounds	176
8	Combining Sounds	181
8.1	Blending Sounds	182
8.2	Creating an Echo	183
8.2.1	Creating Multiple Echoes	183
8.3	How Sampling Keyboards Work	184
8.3.1	Sampling as an algorithm	188
8.4	Additive Synthesis	189
8.4.1	Making sine waves	189
8.4.2	Adding sine waves together	191
8.4.3	Checking our result	193
8.4.4	Square waves	193
8.4.5	Triangle waves	194
8.5	Modern Music Synthesis	195
8.5.1	MP3	196
8.5.2	MIDI	196
9	Design and Debugging	202
9.1	Designing programs: How do we start?	202
9.1.1	Top-down	202
9.1.2	Bottom-up	204
9.2	Techniques of Debugging	205
9.2.1	Seeing the Variables	206
9.3	Algorithms and Design	209
IV	Text, Files, and Unimedia	211
10	Creating and Modifying Text	212
10.1	Strings: Making and manipulating strings	212
10.1.1	String Methods: Introducing objects and dot notation	216
10.1.2	Lists: Powerful, structured text	218
10.1.3	Strings have no font	220
10.2	Files: Places to put your strings and other stuff	221
10.2.1	Opening and manipulating files	223
10.2.2	Writing out programs	224
10.2.3	Copying files	227
10.3	Extending our language capabilities using modules	229
10.3.1	Another fun module: Random	230



	vii
10.4 Networks: Getting our text from the Web	231
10.5 Using text to shift between media	235
10.5.1 Using lists as structured text for media representations	239
11 Making Text for the Web	242
11.1 HTML: The Notation of the Web	242
11.2 Writing programs to generate HTML	246
11.3 Databases: A place to store our text	251
11.3.1 Relational databases	253
11.3.2 Working with SQL	256
11.3.3 Using a database to build Web pages	258
 V Movies	 261
12 Creating and Modifying Movies	262
12.1 Generating Animations	263
12.2 Working with Video	271
12.2.1 Video manipulating examples	271
 VI Topics in Computer Science	 277
13 Speed	278
13.1 What makes programs fast?	278
13.1.1 What computers really understand	278
13.1.2 Compilers and Interpreters	280
13.1.3 How fast can we really go?	283
13.1.4 Making searching faster	285
13.1.5 Algorithms that never finish or can't be written	287
13.1.6 Why is Photoshop faster than JES?	289
14 Styles of Programming	290
14.1 Using functions to make programming easier	290
14.2 Functional programming: Programming in very few lines	293
14.2.1 Recursion: A powerful idea	297
14.3 Object-oriented programming	301
14.3.1 An Object-Oriented Slide Show	302
14.3.2 Object-oriented media	305
14.3.3 Why objects?	310
15 Creating Graphical User Interfaces	312
16 JavaScript	313
16.1 JavaScript syntax	313
16.2 JavaScript inside of Web pages	314
16.3 User interfaces in JavaScript	317
16.4 Multimedia in JavaScript	323





viii

APPENDICES

A Quick Reference to Python	325
A.1 Variables	325





List of Figures

0.1 TO TEACHERS

0.2 TYPOGRAPHICAL NOTATIONS

0.3 ACKNOWLEDGEMENTS

1.1	Eight wires with a pattern of voltages is a byte, which gets interpreted as a pattern of eight 0's and 1's, which gets interpreted as a decimal number.	13
2.1	JES (with annotations)	23
2.2	The File Picker	26
2.3	File picker with media types identified	28
2.4	Picking, making, and showing a picture, using each function as input to the next	29
2.5	Picking, making, and showing a picture, naming the pieces	30
2.6	Visualizing the blocks in JES	34
2.7	Defining and executing <code>pickAndShow</code>	35
3.1	An example matrix	43
3.2	Cursor and icon at regular magnification on top, and close-up views of the cursor (left) and the line below the cursor (right)	44
3.3	Image shown in JES MediaTools picture tool: 100% image on left and 500% on right	45
3.4	Merging red, green, and blue to make new colors	46
3.5	The ends of this figure are the same colors of gray, but the middle two quarters contrast sharply so the left looks darker than the right	46
3.6	The Macintosh OS X RGB color picker	47
3.7	Picking a color using RGB sliders from JES	47
3.8	RGB triplets in a matrix representation	48
3.9	Directly modifying the pixel colors via commands: Note the small yellow line on the left	53
3.10	An example JES Help entry	54
3.11	An example JES Explain entry	54
3.12	Opening a picture in the JES MediaTools picture tool	55
3.13	Using the MediaTools image exploration tools	55
3.14	The original picture (left) and red-reduced version (right)	59
3.15	Using the MediaTools picture tool to convince ourselves that the red was decreased	62
3.16	Overly blue (left) and red increased by 20% (right)	63
3.17	Original (left) and blue erased (right)	64
3.18	Original beach scene (left) and at (fake) sunset (right)	65
3.19	Lightening and darkening of original picture	70
3.20	Negative of the image	71



x LIST OF FIGURES

3.21	Color picture converted to grayscale	72
4.1	Once we pick a mirrorpoint, we can just walk <i>xOffset</i> halfway and subtract/add to mirrorpoint	76
4.2	Original picture (left) and mirrored along the vertical axis (right) . .	77
4.3	Santa mirrored horizontally, bottom to top (left) and top to bottom (right)	78
4.4	Temple of Zeus from the Ancient Agora in Athens, Greece	78
4.5	Coordinates where we need to do the mirroring	79
4.6	The manipulated temple	80
4.7	Copying a picture to a canvas	83
4.8	Copying a picture midway into a canvas	85
4.9	Copying part of a picture onto a canvas	86
4.10	Flowers in the <code>mediasources</code> folder	88
4.11	Collage of flowers	89
4.12	Blending the picture of mom and daughter	94
4.13	Copying a picture to a canvas	95
4.14	Scaling the picture down	97
4.15	Scaling up a picture	98
4.16	Increasing reds in the browns	101
4.17	Increasing reds in the browns, within a certain range	102
4.18	Finding the range of where Jenny’s eyes are red	103
4.19	Finding the range of where Jenny’s eyes are red	103
4.20	Original scene (left) and using our sepia-tone recipe	104
4.21	Reducing the colors (right) from the original (left)	105
4.22	Pictures posterized to two levels (left) and four levels (right)	107
4.23	Making the flower bigger, then blurring to reduce pixellation	108
4.24	Merging red, green, and blue to make new colors	112
4.25	Color: RGB triplets in a matrix representation	113
4.26	Color: The original picture (left) and red-reduced version (right) . .	113
4.27	Color: Overly blue (left) and red increased by 20% (right)	114
4.28	Color: Original (left) and blue erased (right)	114
4.29	Original beach scene (left) and at (fake) sunset (right)	115
4.30	Color: Lightening and darkening of original picture	115
4.31	Color: Negative of the image	116
4.32	Color: Color picture converted to grayscale	116
4.33	Color: Increasing reds in the browns	117
4.34	Color: Increasing reds in the browns, within a certain range	118
4.35	Finding the range of where Jenny’s eyes are red, then changing them to black	118
4.36	Frames from the slow sunset movie	119
4.37	Frames from the slow fade-out movie	119
4.38	Frames from the Mommy watching Katie movie	120
4.39	Frames from the original too dark movie	120
4.40	Frames from the modified lighter movie	121
4.41	Frames from the original movie with kids crawling in front of a blue screen	121
4.42	Frames from the kids on the moon movie	122

LIST OF FIGURES xi

5.1	A picture of a child (Katie), and her background without her	123
5.2	A new background, the moon	124
5.3	Katie on the moon	125
5.4	Two people in front of a wall, and a picture of the wall	125
5.5	Swapping a jungle for the wall, using background subtraction, with a threshold of 50	126
5.6	Mark in front of a blue sheet	126
5.7	Mark on the moon	127
5.8	Mark in the jungle	128
5.9	Student in front of a red background, and with flash on	128
5.10	Using chromakey recipe with red background	129
5.11	A very small, drawn picture	129
5.12	A very small, drawn picture	130
5.13	A programmed gray scale effect	132
5.14	Nested colored rectangles image	133
5.15	Nested blank rectangles image	134
6.1	Raindrops causing ripples in the surface of the water, just as sound causes ripples in the air	138
6.2	One cycle of the simplest sound, a sine wave	139
6.3	The distance between peaks in ripples in a pond are not constant— some gaps are longer, some shorter	140
6.4	The note A above middle C is 440 Hz	141
6.5	Some synthesizers using triangular (or <i>sawtooth</i>) or square waves. . .	142
6.6	Sound editor main tool	142
6.7	Viewing the sound signal as it comes in	143
6.8	Viewing the sound in a spectrum view	144
6.9	Viewing a sound in spectrum view with multiple “spikes”	145
6.10	Viewing the sound signal in a sonogram view	146
6.11	Area under a curve estimated with rectangles	147
6.12	A depiction of the first five elements in a real sound array	148
6.13	A sound recording graphed in the MediaTools	149
6.14	Turning on Expert errors mode	151
6.15	The sound editor open menu	153
6.16	MediaTools open file dialog	153
6.17	A sound opened in the editor	154
6.18	Exploring the sound in the editor	154
6.19	Picking a sound in JES MediaTools	155
6.20	JES MediaTools Sound tool	155
6.21	Zoomed in to the JES MediaTools Sound tool	156
6.22	Comparing the graphs of the original sound (bottom) and the louder one (top)	160
6.23	Comparing specific samples in the original sound (top) and the louder one (bottom)	161
7.1	Comparing the original sound (top) to the spliced sound (bottom) .	175
8.1	The top and middle waves are added together to create the bottom wave	199

xii LIST OF FIGURES

8.2	The raw 440 Hz signal on top, then the 440+880+1320 Hz signal on the bottom	200
8.3	FFT of the 440 Hz sound	200
8.4	FFT of the combined sound	200
8.5	The 440 Hz square wave (top) and additive combination of square waves (bottom)	201
8.6	FFT's of the 440 Hz square wave (top) and additive combination of square waves (bottom)	201
9.1	Seeing the variables using <code>showVars()</code>	207
9.2	Stepping through the <code>makeSunset()</code> function with the Watcher . . .	208
9.3	Watching the variable <code>value</code> in the <code>makeSunset()</code> function with the Watcher	208
10.1	Diagram of a directory tree	222
10.2	Diagram for the tree described in the list	222
10.3	Using Jython outside of JES, even to do media	235
10.4	Sound-as-text file read into Excel	236
10.5	Sound-as-text file graphed in Excel	237
10.6	A visualization of the sound “This is a test”	239
11.1	Simple HTML page source	243
11.2	Simple HTML page open in Internet Explorer	243
11.3	HTML styles	244
11.4	Inserting an image into an HTML page	245
11.5	An HTML page with a link in it	245
11.6	Inserting a table into an HTML page	246
11.7	Creating a thumbnail page	249
11.8	Using the simple database	252
11.9	An example relational table	253
11.10	Representing more complex relationships across multiple tables . . .	254
12.1	MovieMaker application on CD	263
12.2	A few frames from the first movie: Moving a rectangle down and up	264
12.3	Tracing the computation of position	265
12.4	From execution of <code>tickertape</code> function	266
12.5	Moving two rectangles at once	267
12.6	Frames from the slow sunset movie	270
12.7	Frames from the slow fade-out movie	270
12.8	Movie tools in MediaTools	271
12.9	Frames from the Mommy watching Katie movie	272
12.10	Frames from the original kids crawling in front of a blue screen . . .	273
12.11	Frames from the kids on the moon movie	274
12.12	Frames from the original too dark movie	275
12.13	Frames from the new lightened movie	275
13.1	Running the <code>doGraphics</code> interpreter	281
14.1	A hierarchy of functions for creating a samples page	293
14.2	Changing the program is only a slight change to the hierarchy . . .	293
14.3	Examples of rectangle methods	307
14.4	Examples of oval methods	307
14.5	Examples of arc methods	308

LIST OF FIGURES **xiii**

14.6 Examples of text methods	309
16.1 Simple JavaScript function	315
16.2 Showing the parts of the simple JavaScript function	315
16.3 Using JavaScript to insert HTML	316
16.4 Using JavaScript to compute a loop	316
16.5 Computing a list that counts to ten	317
16.6 Inserting the date and time into a web page	318
16.7 Example JavaScript dialog windows	318
16.8 Example catching the onClick event	320
16.9 Opening a JavaScript window	320
16.10 Changing the new JavaScript window	321
16.11 Changing color of list items	322
16.12 A simple HTML form	322
16.13 Inch/centimeter converter in JavaScript	323



xiv LIST OF FIGURES



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.

2 LIST OF FIGURES

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 glass 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.

4 LIST OF FIGURES

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 like this:

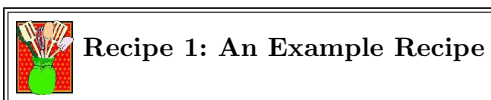
```
def helloWorld():
    print "Hello, world!"
```

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
```

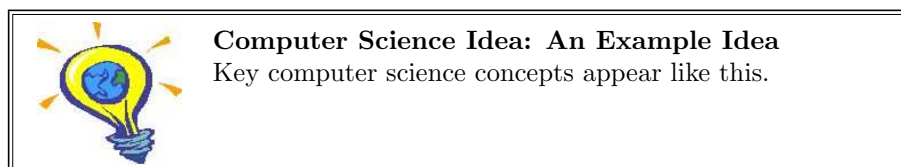
User interface components of JES (Jython Environment for Students) will be specified using a smallcaps font, like SAVE menu item and the LOAD button.

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



Recipes (programs) appear like this:

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





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 Chambliss, 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

6 LIST OF FIGURES

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.