



BRIEF CONTENTS

PREFACE xxi

PART 1 THINKING ABOUT COMPUTING 1

Chapter 0 The Study of Computer Science 3

PART 2 STARTING TO PROGRAM 35

Chapter 1 Beginnings 37

Chapter 2 Control 79

Chapter 3 Algorithms and Program Development 147

PART 3 ORGANIZING: DATA STRUCTURES AND FUNCTIONS 173

Chapter 4 Working with Strings 175

Chapter 5 Functions—QuickStart 227

Chapter 6 Lists and Tuples 251

Chapter 7 More on Functions 313

Chapter 8 Dictionaries and Sets 341

Chapter 9 Files 391

Chapter 10 More Program Development 417

PART 4 CLASSES: MAKING YOUR OWN DATA STRUCTURES &
ALGORITHMS 455

Chapter 11 Introduction to Classes 457

Chapter 12 More on Classes 493

Chapter 13 Program Development with Classes 533

PART 5 BECOMING A BETTER PROGRAMMER 561

Chapter 14 Exceptions and Exception Handling 563

Chapter 15	Testing	583
Chapter 16	Recursion: Another Control Mechanism	601

APPENDICES 619

Appendix A	Getting and Using Python	619
Appendix B	Simple Drawing with Turtle Graphics	631
Appendix C	Plotting and Numeric Tools: A Quick Survey	635
Appendix D	Python 3.0	649
Appendix E	Table of ASCII Values	653
Appendix F	Precedence	655

INDEX 657



CONTENTS

PREFACE xxi

PART 1 THINKING ABOUT COMPUTING 1

Chapter 0 The Study of Computer Science 3

- 0.1 Why Computer Science? 3
 - Importance of Computer Science 3
 - Computer “Science” 4
 - Computer Science Through Computer Programming 5
- 0.2 The Difficulty and Promise of Programming 6
 - Difficulty 1: Two Things at Once 6
 - Difficulty 2: What is a Good Program? 8
 - The Promise of a Computer Program 10
- 0.3 Choosing a Computer Language 10
 - Different Computer Languages 10
 - Why Python? 11
 - Is Python the Best Language? 12
- 0.4 What is Computation? 12
- 0.5 What is a Computer? 13
 - Computation in Nature 13
 - The Human Computer 16
- 0.6 The Modern, Electronic Computer 17
 - It’s the Switch! 17
 - The Transistor 19
- 0.7 A High-Level Look at a Modern Computer 23
- 0.8 Representing Data 25
 - Binary Data 25
 - Working with Binary 26
 - Limits 27

	Representing Letters	28
	Representing Other Data	29
	What Does a Number Represent?	30
	How to Talk About Quantities of Data	31
	How Much Data is That?	31
0.9	Overview of Coming Chapters	32
	Summary	33

PART 2 STARTING TO PROGRAM **35**

Chapter 1 Beginnings 37

1.1	Practice, Practice, Practice	37
1.2	QUICKSTART, the Circumference Program	38
	Examining the Code	40
1.3	An Interactive Session	41
1.4	Parts of a Program	43
	Modules	43
	Statements and Expressions	43
	Whitespace	45
	Comments	46
	Special Python Elements: Tokens	46
	Naming Objects	48
1.5	Variables	49
	Variable Creation and Assignment	50
1.6	Objects and Types	52
	Numbers	55
	Other Built-In Types	57
	Object Types: Not Variable Types	58
	Constructing New Values	59
1.7	Operators	61
	Integer Operators	61
	Floating Point Operators	62
	Mixed Operations	62
	Order of Operations and Parentheses	63
	Augmented Assignment Operators: A Shortcut!	64
1.8	Your First Module, Math	66
1.9	Developing an Algorithm	67
1.10	Conclusion	70

1.11	Visual Vignette: Turtle Graphics	70
	Exercises	72
	Programming Projects	76
Chapter 2	Control	79
2.1	The Selection Statement for Decisions: <i>if</i>	79
	Booleans for Decisions	80
	The <i>if</i> Statement	81
	Example: What Lead is Safe in Basketball?	84
	Repetition	88
	Example: Finding Perfect Numbers	92
	Example: Classifying Numbers	96
2.2	In-Depth Control	99
	True and False: Booleans	99
	Boolean Variables	100
	Relational Operators	100
	Boolean Operators	105
	Precedence	106
	Boolean Operators Example	106
	Another Word on Assignments	110
	The Selection Statement for Decisions	112
	More on Python Decision Statements	113
	Repetition: The <i>while</i> Statement	116
	Sentinel Loop	125
	Summary of Repetition	126
	More on the <i>for</i> Statement	126
	Nesting	129
	Hailstone Sequence Example	129
2.3	Plotting Data with Pylab	132
	First Plot and Using a List	132
	More Interesting Plot: a Sine Wave	134
2.4	Computer Science Perspectives: Minimal Universal Computing	136
	Summary	138
	Exercises	138
	Programming Projects	144
Chapter 3	Algorithms and Program Development	147
3.1	What is an Algorithm?	147
	Example Algorithms	148

- 3.2 Algorithm Features 149
 - Algorithm Versus Program 149
 - Detailed 150
 - Effective 150
 - Specify Behavior 151
 - General-Purpose Algorithms 151
 - Can We Really Do All That? 152
- 3.3 What is a Program? 152
 - Readability 152
 - Robust 155
 - Correctness 156
- 3.4 Strategies for Program Design 157
 - Engage and Commit 157
 - Understand, Then Visualize 158
 - Think Before You Program 159
 - Experiment 159
 - Simplify 160
 - Stop and Think 161
 - Relax: Give Yourself a Break 161
- 3.5 A Simple Example 161
 - Build the Skeleton 162
 - Output 162
 - Input 163
 - Doing the Calculation 166
 - Summary 171
 - Exercises 171

PART 3 ORGANIZING: DATA STRUCTURES AND FUNCTIONS 173

Chapter 4 Working with Strings 175

- 4.1 The String Type 175
 - The Triple Quote String 176
 - Non Printing Characters 177
 - String Representation 177
 - Strings as a Sequence 178
 - More Indexing and Slicing 179
- 4.2 String Operations 183
 - Concatenation (+) and Repetition (*) 184
 - Determining When + Indicates Addition or Concatenation? 185
 - Comparison Operators 186

	The <code>in</code> Operator	187
	String Collections are Immutable	188
4.3	A Preview of Functions and Methods	189
	First Cut: What is a Function?	189
	A String Method	191
	Determining Method Names and Method Arguments	193
	String Methods	195
	String Functions	196
4.4	Formatted Output for Strings	196
	Descriptor Codes	197
	Width Descriptor	198
	Floating-Point Precision Descriptor	199
4.5	Control and Strings	200
4.6	Working with Strings	203
	Example: Reordering a Person's Name	203
	Palindromes	205
4.7	Example: Counting Poker Hands	208
	Program to Count Poker Hands	211
	Summary	218
	Exercises	218
	Programming Projects	222
Chapter 5	Functions—QuickStart	227
5.1	What is a Function?	227
	Why Have Functions?	228
5.2	Python Functions	229
5.3	Flow of Control with Functions	231
	Function Flow in Detail	232
	Another Function Example	235
	Function Example: Word Puzzle	236
	Functions Calling Functions	242
	When to Use a Function	242
	What if There is No Return Statement?	243
	What if There Are Multiple Return Statements?	243
5.4	Visual Vignette: Turtle Flag	244
	Summary	245
	Exercises	245
	Programming Projects	249

Chapter 6	Lists and Tuples	251
6.1	What is a List?	251
6.2	What You Already Know How to Do with Lists	253
	Indexing and Slicing	253
	Operators	254
	Functions	256
	List Iteration	256
6.3	New Things in Lists	256
	Lists are Mutable	256
	List Methods	258
6.4	Old and New Friends: Range, Split, and Other Functions and Methods	261
	Range, Split, and Multiple Assignment	261
	List to String and Back Again, Using <code>join</code>	262
	The Sorted Function	263
6.5	Working with Some Examples	264
	Anagrams	264
	Example: File Analysis	268
6.6	Mutable Objects and References	273
	Shallow vs. Deep Copy	277
	Mutable versus Immutable	280
6.7	Tuples	281
	Tuples from Lists	282
	Why Tuples?	283
6.8	Lists: The Data Structure	283
	Example Data Structure	284
	Other Example Data Structures	285
6.9	Algorithm Example U.S. EPA Automobile Mileage Data	286
6.10	Python Diversion: List Comprehension	296
6.11	Visual Vignette: More Plotting	297
	Numpy Arrays	297
	Plotting Trigonometric Functions	299
	Summary	300
	Exercises	300
	Programming Projects	307
Chapter 7	More on Functions	313
7.1	Functions Calling Functions	313

7.2	Scope: A First Cut	316
	Arguments, Parameters, and Namespaces	317
	Passing Mutable Objects	319
	Returning a Complex Object	321
	Refactoring <code>evens</code>	323
7.3	Default Values and Parameters as Keywords	324
	Example: Default Values and Parameter Keywords	325
	Issues with Default Values	326
7.4	Functions as Objects	328
	Docstrings	328
7.5	Example: Determining a Final Grade	329
	The Data	329
	The Design	329
	Function: <code>weightedGrade</code>	330
	Function: <code>grade</code>	330
	Function: <code>main</code>	331
	Example Use	332
7.6	Esoterica: “by value” or “by reference”	332
	Summary	333
	Exercises	333
	Programming Projects	336
Chapter 8	Dictionaries and Sets	341
8.1	Dictionaries	341
	Dictionary Example	342
	Python Dictionaries	343
	Dictionary Indexing and Assignment	343
	Operators	344
8.2	Word Count Example	348
	Count Words in a String	348
	Word Frequency for Gettysburg Address	349
	Output and Comments	353
8.3	Periodic Table Example	354
	Working with CSV Files	354
	Algorithm Overview	356
	Functions for Divide and Conquer	356
8.4	Sets	361
	History	361
	What’s in a Set?	361
	Python Sets	361

	Methods, Operators, and Functions for Python Sets	362
	Set Methods	363
8.5	Set Applications	367
	Relationship Between Words of Different Documents	367
	Output and Comments	371
8.6	Scope: The Full Story	371
	Namespaces and Scope	372
	Search Rule for Scope	372
	Local	372
	Global	373
	Built-Ins	377
	Enclosed	378
8.7	Python Pointer: Using <code>zip</code> to Create Dictionaries	379
8.8	Visual Vignette: Bar Graph of Word Frequency	380
	Getting the Data Right	381
	Labels and the <code>xticks</code> Command	381
	Plotting	382
	Summary	383
	Exercises	383
	Programming Projects	388
Chapter 9	Files	391
9.1	What is a File?	391
9.2	Accessing Files: Reading Text Files	391
	Other File Access Methods	392
	What's Really Happening?	394
9.3	Accessing Files: Writing Text Files	394
9.4	Reading and Writing Text Files in a Program	395
9.5	File Creation and Overwriting	396
	Universal New Line Format	397
	Moving Around in a File	398
9.6	Closing a File	400
9.7	CSV Files	400
	CSV Module	401
	CSV Reader	402
	CSV Writer	403
	Example: Update Some Grades	403
9.8	Example: Reprompting for a "Good" File Name	405

9.9	Module: os	407
	Directory/Folder Structure	408
	os Module Functions	409
	os Module Example	411
	Summary	413
	Exercises	414
	Programming Projects	415
Chapter 10	More Program Development	417
10.1	Introduction	417
10.2	Divide and Conquer	417
	Top-Down Refinement	418
10.3	The Breast Cancer Classifier	418
	The Problem	418
	The Approach: Classification	419
	Training and Testing the Classifier	419
	Building the Classifier	419
10.4	Designing the Classifier Algorithm	420
	Divided, now Conquer	424
	Data Structures	425
	File Format	425
	Function: makeTrainingSet	425
	The makeTestSet Function	430
	The trainClassifier Function	431
	trainClassifier, Round 2	433
	Testing the Classifier on New Data	436
	The reportResults Function	441
10.5	Running the Classifier on Full Data	442
	Training versus Testing	442
10.6	Other Interesting Problems	446
	Tag Clouds	446
	S&P 500 Predictions	448
	Predicting Religion with Flags	450
	Summary	452
	Exercises	452
	Programming Projects	453

PART 4 CLASSES: MAKING YOUR OWN DATA STRUCTURES & ALGORITHMS **455**

Chapter 11	Introduction to Classes	457
	Simple Student Class	457

- 11.1 Object-Oriented Programming 458
 - Python is Object-Oriented! 458
 - Characteristics of OOP 459
- 11.2 Working with Object-Oriented Programming 459
 - Class and Instance 459
- 11.3 Working with Classes and Instances 460
 - Built-In Class and Instance 460
 - Our First Class 461
 - Changing Attributes 463
 - The Special Relationship Between an Instance and Class:
instance-of 465
- 11.4 Object Methods 467
 - Using Object Methods 467
 - Writing Methods 469
 - The Special Argument `self` 470
 - Methods are the Interface to a Class Instance 472
- 11.5 Fitting into the Python Class Model 472
 - Making Programmer-Defined Classes 472
 - A Student Class 473
 - Python Standard Methods 473
 - Now There Are Three: Class Designer, Programmer,
and User 478
- 11.6 Example: Point Class 478
 - Construction 480
 - Distance 480
 - Summing Two Points 480
 - Improving the Point Class 481
- 11.7 Python and OOP 483
 - Encapsulation 483
 - Inheritance 484
 - Polymorphism 484
- 11.8 An Aside: Python and Other OOP Languages 484
 - Public versus Private 484
 - Indicating Privacy Using Double Underscores (`--`) 485
 - Python's Philosophy 486
 - Modifying an Instance 486
- 11.9 Conclusion 487
 - Exercises 487
 - Programming Projects 488

- Chapter 12 More on Classes 493
 - 12.1 More About Class Properties 493
 - Rational Number (Fraction) Class Example 494
 - 12.2 How Does Python Know? 495
 - Classes, Types, and Introspection 495
 - Remember Operator Overloading 497
 - 12.3 Creating Your Own Operator Overloading 497
 - Mapping Operators to Special Methods 499
 - 12.4 Building the Rational Number Class 501
 - Making the Class 501
 - Review Fraction Addition 503
 - Back to Adding Fractions 506
 - Equality and Reducing Fractions 510
 - Divide and Conquer at Work 513
 - 12.5 What Doesn't Work (Yet) 513
 - Introspection 514
 - Repairing int + Rational Errors 516
 - 12.6 Inheritance 518
 - The "Find the Attribute" Game 518
 - Using Inheritance 521
 - Example: The Standard Model 522
 - Summary 528
 - Exercises 528
- Chapter 13 Program Development with Classes 533
 - 13.1 Predator–Prey Problem 533
 - The Rules 534
 - Simulation Using Object-Oriented Programming 535
 - 13.2 Classes 535
 - Island Class 535
 - Predator and Prey, Kinds of Animals 537
 - Predator and Prey Classes 541
 - Object Diagram 541
 - Filling the Island 541
 - 13.3 Adding Behavior 544
 - Refinement: Add Movement 544
 - Refinement: Time Simulation Loop 547
 - 13.4 Refinement: Eating, Breeding, and Keeping Time 548
 - Improved Time Loop 549
 - Breeding 552

Eating 554
The Tick of the Clock 555

- 13.5 Refinements 556
 - Refinement: How Many Times to Move? 556
 - Refinement: Graphing Population Size 557
- 13.6 Conclusion 559
 - Exercises 559

PART 5 BECOMING A BETTER PROGRAMMER 561

Chapter 14 Exceptions and Exception Handling 563

- 14.1 Introduction 563
- 14.2 Basic Exception Handling 565
 - A Simple Example 566
- 14.3 A Philosophy Concerning Exceptions 569
- 14.4 Exception: `else` and `finally` 570
 - Example: Refactoring the Reprompting of a File Name 570
- 14.5 Exception Usage 572
 - Check Input 572
 - Check File Opening 573
- 14.6 More on Exceptions 574
 - Raise 574
 - Create Your Own 575
- 14.7 Example: Password Manager 576
 - Summary 580
 - Exercises 580

Chapter 15 Testing 583

- 15.1 Why Testing? 583
 - Kinds of Errors 583
 - "Bugs" and Debugging 584
- 15.2 Kinds of Testing 585
 - Testing is Hard! 586
 - Importance of Testing 587
- 15.3 Example Problem 587
 - NBA Efficiency 588
 - Basic Algorithm 588
- 15.4 Incorporating Testing 591
 - Catching User Errors 591
 - Catching Developer Errors 593

15.5	Automation of Testing	595
	doctest	595
	Other Kinds of Testing	598
15.6	Conclusion	598
	Exercises	599
Chapter 16	Recursion: Another Control Mechanism	601
16.1	What is Recursion?	601
16.2	Mathematics and Rabbits	603
16.3	Let's Write Our Own: Reversing a String	606
16.4	How Does Recursion Actually Work?	608
	Stack Data Structure	609
	Stacks and Function Calls	611
16.5	Recursion in Figures	613
	Recursive Tree	613
	Sierpinski Triangles	615
16.6	Recursion to Nonrecursion	616
16.7	Conclusion	617
	Exercises	617
 APPENDICES 619		
Appendix A	Getting and Using Python	619
Appendix B	Simple Drawing with Turtle Graphics	631
Appendix C	Plotting and Numeric Tools: A Quick Survey	635
Appendix D	Python 3.0	649
Appendix E	Table of ASCII Values	653
Appendix F	Precedence	655
 INDEX 657		



P R E F A C E

A FIRST COURSE IN COMPUTER SCIENCE IS ABOUT A NEW WAY OF SOLVING PROBLEMS: computationally. As a result, the focus of a first course in computation should be on problem solving. Practically speaking, however, the teaching of problem solving is inexorably intertwined with the computer language used. Thus, the choice of language for this first course is very important.

We have chosen Python as the introductory language for beginning programming students—majors and nonmajors alike—based on our combined 30 years of experience teaching undergraduate introductory computer science at Michigan State University. Having taught the course in Pascal, C/C++, and now Python, we know that an introductory programming language should have two characteristics. First, it should be relatively simple to learn. Since a first course in programming is mostly about problem solving and learning to be more rigorous in applying problem solving to new problems, the programming language should get in the way as little as possible. Python’s simplicity, powerful built-in data structures, and advanced control constructs allow students to focus more on problem solving and less on language issues. Second, it should be practical. Students should see that they might use what they are learning while they are learning it. Python supports learning not only fundamental programming issues such as typical programming constructs, a fundamental object-oriented approach, common data structures, and so on, but also more complex computing issues such as threads and regular expressions. Python can also be used to address practical problems such as Web access, database manipulation, and thousands of others from nearly any discipline imaginable using tools provided by an active and generous Python community (see <http://pypi.python.org>). From having taken *one* course in programming, a student—any student (major or not)—should walk away from a CS1 course and somewhere later in his or her career address a problem by thinking, “Hey, I’ll just write a program to solve that.” For us, that defines computational thinking. We have seen many examples of this kind of success every semester since the move to Python—in particular, students returning to tell us how useful Python has been to them in their studies and jobs.

We emphasize both the fundamental issues of programming and practicality by focusing on data manipulation and analysis as a theme—allowing students to work on real problems

using either publicly available data sets from various Internet sources or self-generated data sets from their own work and interests. We also emphasize the development of programs, providing multiple, worked out, examples and three entire chapters for detailed design and implementation of programs. As part of this one-semester course, our students have analyzed breast cancer data, catalogued movie actor relationships, predicted disruptions of satellites from solar storms, and completed many other data analysis problems. We have also found that concepts learned in a Python CS1 course transitioned to a CS2 C++ course with little or no impact on either the class material or the students.

Our goals for the book are as follows:

1. Teach problem solving within the context of CS1 to both majors and nonmajors using Python as a vehicle.
2. Provide examples of *developing* programs focusing on the kinds of data analysis problems students might ultimately face.
3. Give students who take no programming course other than this CS1 course a practical foundation in programming, enabling them to produce useful, meaningful results in their respective fields of study.

BOOK ORGANIZATION

At the highest level our text follows a fairly traditional CS1 order, though there are some differences. For example, we cover strings rather early (before functions) so that we can do more data manipulation early on. We also include elementary file I/O early for the same reason, leaving detailed coverage for a later chapter. Given our theme of data manipulation, we feel this is appropriate. We also “sprinkle” topics like plotting and drawing throughout the text in service of the data manipulation theme.

We use an “object-use-first” approach where we use built-in Python objects and their methods early in the book, leaving the design and implementation of user-designed objects for later. We have found that students are more receptive to building their own classes once they have experienced the usefulness of Python’s existing objects. In other words, we motivate the need for writing classes. Functions are split into two parts because of how Python handles mutable objects such as lists as parameters; discussion of those issues can only come after there is an understanding of lists as mutable objects.

Three of the chapters (3, 10, and 13) are primarily program design chapters, providing an opportunity to “tie things together,” as well as showing how to design a solution. A few chapters are intended as supplemental reading material for the students though lecturers may choose to cover these topics as well. For background, we provide a Chapter 0 that introduces some general concepts of a computer as a device and some computer terminology. We feel such an introduction is important—everyone should understand a little about a computer, but this material can be left for reading. The last chapters in the text may not be reached in some courses.

BOOK FEATURES

Data Manipulation

Data manipulation is a theme. The examples range from text analysis to breast cancer classification. Along the way, we provide some analysis examples using simple graphing. To incorporate drawing and graphing, we use established packages instead of developing our own: one is built in (Turtle graphics); the other is widely used (matplotlib with numpy).

We have tried to focus on non-numeric examples in the book, but some numeric examples are classics for a good reason. For example, we use a rational numbers example for creating classes that overload operators. Our goal is always to use the best examples.

Problem Solving and Case Studies

Throughout the text, we emphasize problem solving, especially a divide-and-conquer approach to developing a solution. Three chapters (3, 10, and 13) are devoted almost exclusively to program development. Here we walk students through the solution of larger examples. In addition to design, we show mistakes and how to recover from them. That is, we don't simply show a solution, but show a *process of developing* a solution.

Code examples

There are over 180 code examples in the text—many are brief, but others illustrate piecemeal development of larger problems.

Interactive Sessions

The Python interpreter provides a wonderful mechanism for briefly illustrating programming and problem-solving concepts. We provide almost 250 interactive sessions for illustration.

Exercises and Programming Projects

Practice, practice, and more practice. We provide over 275 short exercises for students and nearly 30 longer programming projects (many with multiple parts).

Self-Test Exercises

Embedded within the chapters are 24 self-check exercises, each with 5 or more associated questions.

Programming Tips

We provide over 40 special notes to students on useful tips and things to watch out for. These tips are boxed for emphasis.

SUPPLEMENTARY MATERIAL ONLINE

- For students
 - All example source code
 - Data sets used in examples

The above material is freely available at www.pearsonhighered.com/cssupport.

- For instructors
 - PowerPoint slides
 - Laboratory exercises
 - Figures (PDF) for use in your own slides
 - Exercise solutions

Qualified instructors may obtain supplementary material by visiting www.pearsonhighered.com/irc. Register at the site for access. You may also contact your local Pearson Education representative or send e-mail to computing@aw.com.

Python 2.x versus 3.x

The text is based on Python 2.x (for $x \geq 6$) rather than Python 3.x since the latter is not backward compatible with standard Python packages at this time. In an appendix we outline the differences. For what we cover in the text, the differences are trivial, but the support of current Python packages is critical to our approach, and we will not be able to move to 3.x until standard packages are more widely available in that release.

ACKNOWLEDGEMENTS

We are particularly appreciative of the reviewers who provided valuable feedback at multiple stages of this endeavor: Claude Anderson (Rose-Hulman Institute of Technology), Chris Brooks (University of San Francisco), Judy Franklin (Smith College), Alan Garvey (Truman State University), Ronald I. Greenberg (Loyola University), Andrew Harrington (Loyola College), Steve Harrison (Virginia Tech), Christopher Haynes (Indiana University), Cinda Heeren (University of Illinois/Urbana-Champaign), Brian Howard (DePauw University), Janardhan Iyengar (Franklin & Marshall College), Andree Jacobson (University of New Mexico), John Lasseter (Willamette University), Jim Mahoney (Marlboro College), Joe Oldham (Centre College), Holly Patterson-McNeill (Lewis Clark State College), John

Rabung (Randolph-Macon College), Ben Schafer (University of Northern Iowa), David G. Sullivan (Boston University), David A. Sykes (Wofford College). Here at Michigan State University, Erik Eid (now of Bowling Green State University) provided valuable feedback on the first draft. Laurie Dillon provided feedback when she taught from a draft. C. Titus Brown read a draft from a Pythonic perspective and provided encouragement. As a high school student, Angus Burton provided valuable feedback from a novice's perspective. Srikanth Vudayagiri provided many excellent exercises. Scott Buffa made corrections to an early draft. The summer CSE 231 class provided many exercises. Members of the class were: Mohammed Alwahibi, Younsuk Dong, Michael Ford, Gabriel Friedman, Adam Hamilton, Meagan Houang, Ruba Jiddou, and Adam Palmer. The organization of our course was established by Mark McCullen, who is always available for advice. Larry Nyhoff (Calvin College) shared valuable insights over a dinner that started it all.

W. F. PUNCH
R. J. ENBODY

