

Summary of Contents



List of Figures	xix
Acknowledgments	xxi
0 Introduction	1
1 Unix	12
2 Version Control	55
3 Basic Programming	81
4 Writing Good Code	120
5 Regular Expressions	165
6 Scientific Computing	185
7 Scientific Typesetting	220
8 Statistical Computing	249
9 Data Wrangling and Visualization	300
10 Relational Databases	337
11 Wrapping Up	366
Intermezzo Solutions	373
Bibliography	389
Indexes	393

© Copyright Princeton University Press. No part of this book may be distributed, posted, or reproduced in any form by digital or mechanical means without prior written permission of the publisher.

For general queries contact webmaster@press.princeton.edu.

Contents



List of Figures	xix
Acknowledgments	xxi
0 Introduction: Building a Computing Toolbox	1
0.1 The Philosophy	2
0.2 The Structure of the Book	4
0.2.1 How to Read the Book	6
0.2.2 Exercises and Further Reading	6
0.3 Use in the Classroom	8
0.4 Formatting of the Book	10
0.5 Setup	10
1 Unix	12
1.1 What Is Unix?	12
1.2 Why Use Unix and the Shell?	13
1.3 Getting Started with Unix	14
1.3.1 Installation	14
1.3.2 Directory Structure	15
1.4 Getting Started with the Shell	17
1.4.1 Invoking and Controlling Basic Unix Commands	18
1.4.2 How to Get Help in Unix	19
1.4.3 Navigating the Directory System	20
1.5 Basic Unix Commands	22
1.5.1 Handling Directories and Files	22
1.5.2 Viewing and Processing Text Files	24
1.6 Advanced Unix Commands	27
1.6.1 Redirection and Pipes	27
1.6.2 Selecting Columns Using cut	29
1.6.3 Substituting Characters Using tr	32

1.6.4	Wildcards	35
1.6.5	Selecting Lines Using <code>grep</code>	36
1.6.6	Finding Files with <code>find</code>	39
1.6.7	Permissions	41
1.7	Basic Scripting	43
1.8	Simple for Loops	47
1.9	Tips, Tricks, and Going beyond the Basics	49
1.9.1	Setting a <code>PATH</code> in <code>.bash_profile</code>	49
1.9.2	Line Terminators	50
1.9.3	Miscellaneous Commands	50
1.10	Exercises	51
1.10.1	Next Generation Sequencing Data	51
1.10.2	Hormone Levels in Baboons	51
1.10.3	Plant–Pollinator Networks	52
1.10.4	Data Explorer	53
1.11	References and Reading	53
2	Version Control	55
2.1	What Is Version Control?	55
2.2	Why Use Version Control?	55
2.3	Getting Started with Git	56
2.3.1	Installing Git	57
2.3.2	Configuring Git after Installation	57
2.3.3	How to Get Help in Git	58
2.4	Everyday Git	58
2.4.1	Workflow	58
2.4.2	Showing Changes	64
2.4.3	Ignoring Files and Directories	65
2.4.4	Moving and Removing Files	66
2.4.5	Troubleshooting Git	66
2.5	Remote Repositories	68
2.6	Branching and Merging	70
2.7	Contributing to Public Repositories	78
2.8	References and Reading	79
3	Basic Programming	81
3.1	Why Programming?	81
3.2	Choosing a Programming Language	81
3.3	Getting Started with Python	83

3.3.1	Installing Python and Jupyter	83
3.3.2	How to Get Help in Python	84
3.3.3	Simple Calculations with Basic Data Types	85
3.3.4	Variable Assignment	87
3.3.5	Built-In Functions	89
3.3.6	Strings	90
3.4	Data Structures	93
3.4.1	Lists	93
3.4.2	Dictionaries	96
3.4.3	Tuples	100
3.4.4	Sets	101
3.5	Common, General Functions	103
3.6	The Flow of a Program	105
3.6.1	Conditional Branching	105
3.6.2	Looping	107
3.7	Working with Files	112
3.7.1	Text Files	112
3.7.2	Character-Delimited Files	115
3.8	Exercises	117
3.8.1	Measles Time Series	117
3.8.2	Red Queen in Fruit Flies	118
3.9	References and Reading	118
4	Writing Good Code	120
4.1	Writing Code for Science	120
4.2	Modules and Program Structure	121
4.2.1	Writing Functions	121
4.2.2	Importing Packages and Modules	126
4.2.3	Program Structure	127
4.3	Writing Style	133
4.4	Python from the Command Line	135
4.5	Errors and Exceptions	137
4.5.1	Handling Exceptions	138
4.6	Debugging	139
4.7	Unit Testing	146
4.7.1	Writing the Tests	147
4.7.2	Executing the Tests	149
4.7.3	Handling More Complex Tests	150

4.8	Profiling	153
4.9	Beyond the Basics	155
4.9.1	Arithmetic of Data Structures	155
4.9.2	Mutable and Immutable Types	156
4.9.3	Copying Objects	158
4.9.4	Variable Scope	160
4.10	Exercises	161
4.10.1	Assortative Mating in Animals	161
4.10.2	Human Intestinal Ecosystems	162
4.11	References and Reading	163
5	Regular Expressions	165
5.1	What Are Regular Expressions?	165
5.2	Why Use Regular Expressions?	165
5.3	Regular Expressions in Python	166
5.3.1	The <code>re</code> Module in Python	166
5.4	Building Regular Expressions	167
5.4.1	Literal Characters	168
5.4.2	Metacharacters	168
5.4.3	Sets	169
5.4.4	Quantifiers	170
5.4.5	Anchors	171
5.4.6	Alternations	172
5.4.7	Raw String Notation and Escaping Metacharacters	173
5.5	Functions of the <code>re</code> Module	175
5.6	Groups in Regular Expressions	179
5.7	Verbose Regular Expressions	181
5.8	The Quest for the Perfect Regular Expression	181
5.9	Exercises	182
5.9.1	Bee Checklist	182
5.9.2	A Map of <i>Science</i>	182
5.10	References and Reading	184
6	Scientific Computing	185
6.1	Programming for Science	185
6.1.1	Installing the Packages	185

6.2	Scientific Programming with NumPy and SciPy	185
6.2.1	NumPy Arrays	186
6.2.2	Random Numbers and Distributions	194
6.2.3	Linear Algebra	196
6.2.4	Integration and Differential Equations	197
6.2.5	Optimization	200
6.3	Working with pandas	202
6.4	Biopython	208
6.4.1	Retrieving Sequences from NCBI	208
6.4.2	Input and Output of Sequence Data Using SeqIO	210
6.4.3	Programmatic BLAST Search	212
6.4.4	Querying PubMed for Scientific Literature Information	214
6.5	Other Scientific Python Modules	216
6.6	Exercises	216
6.6.1	Lord of the Fruit Flies	216
6.6.2	Number of Reviewers and Rejection Rate	217
6.6.3	The Evolution of Cooperation	217
6.7	References and Reading	219
7	Scientific Typesetting	220
7.1	What Is L ^A T _E X?	220
7.2	Why Use L ^A T _E X?	220
7.3	Installing L ^A T _E X	223
7.4	The Structure of L ^A T _E X Documents	223
7.4.1	Document Classes	224
7.4.2	L ^A T _E X Packages	224
7.4.3	The Main Body	225
7.4.4	Document Sections	227
7.5	Typesetting Text with L ^A T _E X	228
7.5.1	Spaces, New Lines, and Special Characters	228
7.5.2	Commands and Environments	228
7.5.3	Typesetting Math	229
7.5.4	Comments	231
7.5.5	Justification and Alignment	232
7.5.6	Long Documents	232
7.5.7	Typesetting Tables	233
7.5.8	Typesetting Matrices	236

7.5.9	Figures	237
7.5.10	Labels and Cross-References	240
7.5.11	Itemized and Numbered Lists	241
7.5.12	Font Styles	241
7.5.13	Bibliography	242
7.6	\LaTeX Packages for Biologists	244
7.6.1	Sequence Alignments with \LaTeX	245
7.6.2	Creating Chemical Structures with \LaTeX	246
7.7	Exercises	246
7.7.1	Typesetting Your Curriculum Vitae	246
7.8	References and Reading	247
8	Statistical Computing	249
8.1	Why Statistical Computing?	249
8.2	What Is R?	249
8.3	Installing R and RStudio	250
8.4	Why Use R and RStudio?	250
8.5	Finding Help	251
8.6	Getting Started with R	251
8.7	Assignment and Data Types	253
8.8	Data Structures	255
8.8.1	Vectors	255
8.8.2	Matrices	257
8.8.3	Lists	261
8.8.4	Strings	262
8.8.5	Data Frames	263
8.9	Reading and Writing Data	264
8.10	Statistical Computing Using Scripts	267
8.10.1	Why Write a Script?	267
8.10.2	Writing Good Code	267
8.11	The Flow of the Program	270
8.11.1	Branching	270
8.11.2	Loops	272
8.12	Functions	275
8.13	Importing Libraries	278
8.14	Random Numbers	279
8.15	Vectorize It!	280
8.16	Debugging	283
8.17	Interfacing with the Operating System	284

8.18	Running R from the Command Line	285
8.19	Statistics in R	287
8.20	Basic Plotting	290
8.20.1	Scatter Plots	290
8.20.2	Histograms	291
8.20.3	Bar Plots	292
8.20.4	Box Plots	292
8.20.5	3D Plotting (in 2D)	293
8.21	Finding Packages for Biological Research	293
8.22	Documenting Code	294
8.23	Exercises	295
8.23.1	Self-Incompatibility in Plants	295
8.23.2	Body Mass of Mammals	296
8.23.3	Leaf Area Using Image Processing	296
8.23.4	Titles and Citations	297
8.24	References and Reading	297
9	Data Wrangling and Visualization	300
9.1	Efficient Data Analysis and Visualization	300
9.2	Welcome to the tidyverse	300
9.2.1	Reading Data	301
9.2.2	Tibbles	302
9.3	Selecting and Manipulating Data	304
9.3.1	Subsetting Data	305
9.3.2	Pipelines	307
9.3.3	Renaming Columns	308
9.3.4	Adding Variables	309
9.4	Counting and Computing Statistics	310
9.4.1	Summarize Data	310
9.4.2	Grouping Data	310
9.5	Data Wrangling	313
9.5.1	Gathering	313
9.5.2	Spreading	315
9.5.3	Joining Tibbles	316
9.6	Data Visualization	318
9.6.1	Philosophy of ggplot2	319
9.6.2	The Structure of a Plot	320
9.6.3	Plotting Frequency Distribution of One Continuous Variable	321

9.6.4	Box Plots and Violin Plots	322
9.6.5	Bar Plots	323
9.6.6	Scatter Plots	324
9.6.7	Plotting Experimental Errors	325
9.6.8	Scales	326
9.6.9	Faceting	328
9.6.10	Labels	329
9.6.11	Legends	330
9.6.12	Themes	331
9.6.13	Setting a Feature	332
9.6.14	Saving	332
9.7	Tips & Tricks	333
9.8	Exercises	335
9.8.1	Life History in Songbirds	335
9.8.2	Drosophilidae Wings	335
9.8.3	Extinction Risk Meta-Analysis	335
9.9	References and Reading	336
10	Relational Databases	337
10.1	What Is a Relational Database?	337
10.2	Why Use a Relational Database?	338
10.3	Structure of Relational Databases	340
10.4	Relational Database Management Systems	341
10.4.1	Installing SQLite	341
10.4.2	Running the SQLite RDBMS	341
10.5	Getting Started with SQLite	342
10.5.1	Comments	342
10.5.2	Data Types	342
10.5.3	Creating and Importing Tables	343
10.5.4	Basic Queries	344
10.6	Designing Databases	352
10.7	Working with Databases	355
10.7.1	Joining Tables	355
10.7.2	Views	358
10.7.3	Backing Up and Restoring a Database	359
10.7.4	Inserting, Updating, and Deleting Records	360
10.7.5	Exporting Tables and Views	361
10.8	Scripting	362
10.9	Graphical User Interfaces (GUIs)	362

10.10	Accessing Databases Programmatically	362
10.10.1	In Python	363
10.10.2	In R	363
10.11	Exercises	364
10.11.1	Species Richness of Birds in Wetlands	364
10.11.2	Gut Microbiome of Termites	364
10.12	References and Reading	365
11	Wrapping Up	366
11.1	How to Be a More Efficient Computational Biologist	367
11.2	What Next?	368
11.3	Conclusion	371
	Intermezzo Solutions	373
	Bibliography	389
	Indexes	393
	Index of Symbols	393
	Index of Unix Commands	395
	Index of Git Commands	397
	Index of Python Functions, Methods, Properties, and Libraries	399
	Index of \LaTeX Commands and Libraries	401
	Index of R Functions and Libraries	403
	Index of SQLite Commands	405
	General Index	407

CHAPTER 0



Introduction: Building a Computing Toolbox

No matter how much time you spend in the field or at the bench, most of your research is done when sitting in front of a computer. Yet, the typical curriculum of a biology PhD does not include much training on how to use these machines. It is assumed that students will figure things out by themselves, unless they join a laboratory devoted to computational biology—in which case they will likely be trained by other members of the group in the laboratory’s (often idiosyncratic) selection of software tools. But for the vast majority of academic biologists, these skills are learned the hard way—through painful trial and error, or during long sessions sitting with the one student in the program who is “good with computers.”

This state of affairs is at odds with the enormous growth in the size and complexity of data sets, as well as the level of sophistication of the statistical and mathematical analysis that goes into a modern scientific publication in biology. If, once upon a time, coming up with an original idea and collecting great data meant having most of the project ready, today the data and ideas are but the beginning of a long process, culminating in publication.

The goal of this book is to build a basic computational toolbox for biologists, useful both for those doing laboratory and field work, and for those with a computational focus. We explore a variety of tools and show how they can be integrated to construct complex pipelines for automating data collection, storage, analysis, visualization, and the preparation of manuscripts ready for submission.

These tools are quite disparate and can be thought of as LEGO[®] bricks, that can be combined in new and creative ways. Once you have added a new tool to your toolbox, the potential for new research is greatly expanded. Not only will you be able to complete your tasks in a more organized, efficient, and reproducible way, but you will attempt answering new questions that would have been impossible to tackle otherwise.

0.1 The Philosophy

Fundamentally, this book is a manifesto for a certain approach to computing in biology. Here are the main points we want to emphasize:

Automation

Doing science involves repeating the same tasks several times. For example, you might need to repeat an analysis when new data are added, or if the same analysis needs to be carried out on separate data sets, or again if the reviewers ask you to change this or that part of the analysis to make sure that the results are robust.

In all of these cases you would like to automate the processing of the data, such that the data organization and analysis and the production of figures and statistical results can be repeated without any effort. Throughout the book, we keep automation at the center of our approach.

Reproducibility

Science should be reproducible, and much discussion and attention goes into carefully documenting empirical experiments so that they can be repeated. In theory, reproducing statistical analysis or simulations should be much easier, provided that the data and parameters are available. Yet, this is rarely the case—especially when the processing of the data involves clicking one’s way through a graphical interface without documenting all the steps. In order to make it easy to reproduce your results, your computational work should be

readable: Your analysis should be easy to read and understand. This involves writing good code and documenting what you are doing. The best way to proceed is to think of your favorite reader: yourself, six months from now. When you receive feedback from the reviewers, and you have to modify the analysis, will you be able to understand precisely what you did, how, and why? Note that there is no way to email yourself in the past to ask for clarifications.

organized: Keeping the project tidy and well organized is a struggle, but you don’t want to open your project directory only to find that there are 16 versions of the same program, all with slight—and undocumented—variations!

self-contained: Ideally, you want all of your data, code, and results in the same place, without dependencies on other files or code that are not in the same location. In this way, it is easy to share your work with others, or to work on your projects from different computers.

Openness

Science is a worldwide endeavor. If you use costly, proprietary software, the chances are that researchers in less fortunate situations cannot reproduce your results or use your methods to analyze their data. Throughout the book, we focus on *free software*:¹ not only is the software free in the sense that it costs nothing, but free also means that you have the *freedom* to run, copy, distribute, study, change, and improve the software.

Simplicity

Try to keep your analysis as simple as possible. Sometimes, “readable” and “clever” are at odds, meaning that a single line of code processing data in 14 different ways at once might be genius, but seldom is it going to be readable. In such cases, we tend to side with readability and simplicity—even if this means writing three additional lines of code. We also advocate the use of plain text whenever possible, as text is portable to all computer architectures and will be readable decades from now.

Correctness

Your analysis should be correct. This means that programming in science is very different from programming in other areas. For example, *bugs* (errors in the code) are something the software industry has learned to manage and live with—if your application unexpectedly closes or if your word processor sometimes goes awry, it is surely annoying, but unless you are selling pacemakers this is not going to be a threat. In science, it is essential that your code does solely what it is meant to do: otherwise your results might be unjustified. This strong emphasis on correctness is peculiar to science, and therefore you

1. gnu.org/philosophy/free-sw.html.

will not find all of the material we present in a typical programming textbook. We explore basic techniques meant to ensure that your code is correct and we encourage you to rewrite the same analysis in (very) different programming languages, forcing you to solve the problem in different ways; if all programs yield exactly the same results, then they are probably correct.

Science as Software Development

There is a striking parallel between the process of developing software and that of producing science. In fact, we believe that basic tools adopted by software developers (such as version control) can naturally be adapted to the world of research. We want to build software pipelines that turn ideas and data into published work; the development of such a pipeline has important milestones, which parallel those of software development: one can think of a manuscript as a “beta version” of a paper, and even treat the comments of the reviewers as bugs in the project which we need to fix before releasing our product! The development of these pipelines is another central piece of our approach.

0.2 The Structure of the Book

The book is composed of 10 semi-independent chapters:

Chapter 1: Unix

We introduce the Unix command line and show how it can be used to automate repetitive tasks and “massage” your data prior to analysis.

Chapter 2: Version control

Version control is a way to keep your scientific projects tidily organized, collaborate on science, and have the whole history of each project at your fingertips. We introduce this topic using Git.

Chapter 3: Basic programming

We start programming, using Python as an example. We cover the basics: from assignments and data structures to the reading and writing of files.

Chapter 4: Writing good code

When we write code for science, it has to be correct. We show how to organize your code in an effective way, and introduce debugging, unit testing, and profiling, again using Python.

Chapter 5: Regular expressions

When working with text, we often need to find snippets of text matching a certain “pattern.” Regular expressions allow you to describe to a computer what you are looking for. We show how to use the Python module `re` to extract information from text.

Chapter 6: Scientific computing

Modern programming languages offer specific libraries and packages for performing statistics, simulations, and implementing mathematical models. We briefly cover these tools using Python. In addition, we introduce Biopython, which facilitates programming for molecular biology.

Chapter 7: Scientific typesetting

We introduce \LaTeX for scientific typesetting of manuscripts, theses, and books.

Chapter 8: Statistical computing

We introduce the statistical software R, which is fully programmable and for which thousands of packages written by scientists for scientists are available.

Chapter 9: Data wrangling and visualization

We introduce the `tidyverse`, a set of R packages that allow you to write pipelines for the organization and analysis of large data sets. We also show how to produce beautiful figures using `ggplot2`.

Chapter 10: Relational Databases

We present relational databases and `sqlite3` for storing and working efficiently with large amounts of data.

Clearly, there is no way to teach these computational tools in 10 brief chapters. In fact, in your library you will find several thick books devoted to each and every one of the tools we are going to explore. Similarly, becoming a proficient programmer cannot be accomplished by reading a few pages, but rather it requires hundreds of hours of practice. So why try to cover so much material instead of concentrating on a few basic tools?

The idea is to provide a structured guide to help jump-start your learning process for each of these tools. This means that we emphasize breadth over depth (a very unusual thing to do in academia!) and that success strongly depends on your willingness to practice by trying your hand at the exercises and embedding these tools in your daily work. Our goal is to *showcase* each tool by first explaining what the tool is and why you should master it. This allows you to make an informed decision on whether to invest your time in learning how to use it. We then guide you through some basic

features and give you a step-by-step explanation of several simple examples. Once you have worked through these examples, the learning curve will appear less steep, allowing you to find your own path toward mastering the material.

0.2.1 How to Read the Book

We have written the book such that it can be read in the traditional way: start from the first page and work your way toward the end. However, we have striven to provide a modular structure, so that you can decide to skip some chapters, focus on only a few, or use the book as a quick reference.

In particular, the chapters on Unix (ch. 1), version control (ch. 2), \LaTeX (ch. 7), and databases (ch. 10) can be read quite independently: you will sometimes find references to other chapters, but in practice there are no prerequisites. Also, you can decide to skip any of these chapters (though we love each of these tools!) without affecting the reading of the rest of the book.

We present programming in Python (chs. 3–6) and then again in R (chs. 8–9). While we go into more detail when explaining basic concepts in Python, you should be able to understand all of the R material without having read any of the other chapters. Similarly, if you do not plan to use R, you can skip these chapters without impacting the rest of the book.

0.2.2 Exercises and Further Reading

In each chapter, upon completion of the material you will be ready to start working on the “Exercises” section. One of the main features of this book is that exercises are based on real biological data taken from published papers. As such, these are not silly little exercises, but rather examples of the challenges you will overcome when doing research. We have seen that some students find this level of difficulty frustrating. It is entirely normal, however, to have no idea how to solve a problem at first. Whenever you feel that frustration is blocking your creativity and efficiency, take a short break. When you return, try breaking the problem into smaller steps, or start from a blank slate and attempt an entirely different approach. If you keep chipping away at the exercise, then little by little you will make sense of what the problem entails and—finally—you will find a way to crack it. Learning how to enjoy problem solving and to take pride in a job well done are some of the main characteristics of a good scientist.

For example, we are fond of this quote from Andrew Wiles (who proved the famous Fermat's last theorem, which baffled mathematicians for centuries): "You enter the first room of the mansion and it's completely dark. You stumble around bumping into the furniture but gradually you learn where each piece of furniture is. Finally, after six months or so, you find the light switch, you turn it on, and suddenly it's all illuminated."² Hopefully, it will take you less than six months to crack the exercises!

Note that there isn't "a" way to solve a problem, but rather a multitude of (roughly) equivalent ways to do so. Each and every approach is perfect, provided that the results are correct and that the solution is found in a reasonable amount of time. Thus, we encourage you to consult our solutions to the exercises only once you have solved them: Did we come up with the same idea? What are the advantages and disadvantages of these approaches? Even if you did not solve the task entirely, you have likely learned a lot more while trying, compared to reading through the solutions upon hitting the first stumbling block. To provide a further stepping stone between having no idea where to start and a complete solution, we provide a pseudocode solution of each exercise online: the individual steps of the solution are described in English, but no code is provided. This will give you an idea how to approach the problem, but you will need to come up with the code. From there, it is only a short way to tackling your very own research questions. You can find the complete solutions and the pseudocode at computingskillsforbiologists.com/exercises.

When solving the exercises, the internet is your friend. Finding help online is by no means considered "cheating." On the contrary, if you find yourself exploring additional resources, you are doing exactly the right thing! As with research, anything goes, as long as you can solve your problem (and give credit where credit is due). Consulting the many comprehensive online forums gives you a sense of how widespread these computational tools are. Keep in mind that the people finding clever answers to your questions also started from a blank slate at some point in their career. Moreover, seeing that somebody else asked exactly your question should further convince you that you are on the right track.

Last but not least, the "Reading" section of each chapter contains references to books, tutorials, and online resources to further the knowledge of the material. If the chapter is an appetizer, meant to whet your appetite for knowledge, the actual meal is contained in the reading list. This book is a road map that equips you with sufficient knowledge to choose the appropriate tool for each task, and take the guesswork out of "Where should I start

2. computingskillsforbiologists.com/provingfermat.

my learning journey?” However, only by reading more on the topic and by introducing these tools into your daily research work will you be able to truly master these skills, and make the most of your computer.

We conclude with the sales pitch we use to present the class that inspired this book. If you are a graduate student and you read the material, you work your way through all the exercises, constantly striving to further your knowledge of these topics by introducing them into your daily work, then you will shave six months off your PhD—and not *any* six months, but rather those spent wrestling with the data, repeating tedious tasks, and trying to convince the computer to be reasonable and spit out your thesis. All things considered, this book aims to make you a happier, more productive, and more creative scientist. Happy computing!

0.3 Use in the Classroom

We have been teaching the material covered in this book, in the graduate class Introduction to Scientific Computing for Biologists, at the University of Chicago since 2012. The enrollment has been about 30 students per year. We found the material appropriate for junior graduate students as well as senior undergraduates with some research experience.

The University of Chicago runs on a quarter system, allowing for 10 lectures of three hours each. Typically, each chapter is covered by a single lecture, with “Version Control” (ch. 2) and “Scientific Typesetting” (ch. 7) each taking about an hour and a half, and “Writing Good Code” (ch. 4) and “Statistical Computing” (ch. 8) taking more than one lecture each.

In all cases, we taught students who had computers available in class, either by teaching in a computer lab, or by asking students to bring their personal laptops. Rather than using slides, the instructor lectured while typing all the code contained in the book during the class. This makes for a very interactive class, in which all students type all of the code too—making sure that they understand what they are doing. Clearly, this also means that the pace is slowed down every time a student has included a typo in their commands, or cannot access their programs. To ease this problem, having teaching assistants for the class helps immensely. Students can raise their hand, or stick a red post-it on their computer to signal a problem. The teaching assistant can immediately help the student and interrupt the class in case the problem is shared by multiple students—signaling the need for a more general explanation.

To allow the class to run smoothly, each student should prepare their computer in advance. We typically circulate each chapter a week in advance of class, encouraging the students to (a) install the software needed for the class

and (b) read the material beforehand. Teaching assistants also offer weekly office hours to help with the installation of software, or to discuss the material and the exercises in small groups.

The “intermezzos” that are interspersed in each chapter function very well as small in-class exercises, allowing the students to solidify their knowledge, as well as highlighting potential problems with their understanding of the material.

We encourage the students to work in groups on the exercises at the end of each chapter, and review the solutions at the beginning of the following class. While this can cause some difficulties in grading, we believe that working in groups is essential to overcome the challenge of the exercises, making the students more productive, and allowing less experienced students to learn from their peers. Publishing a blog where each group posts their solutions reinforces the esprit de corps, creating a healthy competition between the groups, and further instilling in the students a sense of pride for a job well done. We also encouraged students to constructively comment on the different approaches of other groups and discuss the challenges they’ve faced while solving the exercises.

Another characteristic of our class has been the emphasis on the practical value of the material. For example, we ask each student to produce a final project in which they take a boring, time-consuming task in their laboratory (e.g., analysis of batches of data produced by laboratory machines, calibration of methods, other repetitive computational tasks) and completely automate it. The student then shows their work to their labmates and scientific advisor, and writes a short description of the program, along with the documentation necessary to use it. The goal of the final project is simply to show the student that mastering this material can save them a lot of time—even when accounting for the strenuous process of writing their first programs.

We have also experimented with a “flipped classroom” setting, with mixed results. In this case, the students read the material at their own pace, and work through all the small exercises contained in the chapter. The lecture is then devoted to working on the exercises at the end of each chapter. The lecturer guides the discussion on the strategies that can be employed to solve the problem, sketching pseudocode on the board, and eventually producing a fully fledged code on the computer. We have observed that, while this approach is very rewarding for students with some prior experience in programming, it is much less engaging for novices, who feel lost and out of touch with the rest of the class. Probably, this would work much better if the class size were small (less than 10 students).

Finally, we have found that leading by example serves as powerful motivation to students. We have always shown that we use the tools covered here for our own research. A well-placed anecdote on Git saving the day, or showing

how all the tables in a paper were automatically generated with a few lines of R, can go a long way toward convincing the students that their work studying the material will pay off over a lifetime.

0.4 Formatting of the Book

You will find all commands and the names of packages typeset in a fixed-width font. User-provided [INPUT] is capitalized and set between square brackets. To execute the commands, you do not need to reproduce such formatting. Within explanatory text, *technical terms* are presented in italics.

Throughout the book, we provide many code examples, enclosed in gray boxes and typeset using fixed-width fonts. All code examples are also provided on the companion website computingskillsforbiologists.com—but we encourage you to type all the code in by yourself: while this might feel slow and inefficient, the learning effect is stronger compared to simply copying and pasting, and only inspecting the result. Within the code examples, language-specific commands are highlighted in bold.

Within the code boxes, we try to keep lines short. When we cannot avoid a line that is longer than the width of the page we use the symbol ↪ to indicate that what follows should be typed in the same line as the rest.

0.5 Setup

Before you can start computing, you need to set up the environment, and download the data and the code.

What You Need

A computer: All the software we present here is free and can be installed with a few commands in Linux Ubuntu or Apple's OS X; we strive to provide guidance for Windows users. There are no specific hardware requirements. All the tools require relatively little memory and space on your hard drive.

Software: Each chapter requires installing specific software. We have collected detailed instructions guiding you through the installation of each tool at computingskillsforbiologists.com/setup.

A text editor: While working through the chapters, you will write a lot of code. Much will be written in the integrated development environments (IDEs) Jupyter and RStudio. Sometimes, however, you will need to write code in a text editor. We

encourage you to keep working with your favorite editor, if you already have one. If not, please choose an editor that can support syntax highlighting for Python, R, and \LaTeX . There are many options to choose from, depending on your architecture and needs.³

Initial Setup

You can find instructions for the initial setup on our website at computingskillsforbiologists.com/setup. We have bundled all the data, code, exercises, and solutions in a single download. We strongly recommend that you save this directory in your home directory (see section 1.3.2).

3. computingskillsforbiologists.com/texteditors.

General Index



Italic pages refer to figures and tables

- absolute path, 21–22, 40
- algorithms, 185, 195, 370
- alternations, 172
- amino acids, 90, 145–46, 245
- Anaconda, 84, 176n1
- anchors, 171–72
- annealing, 370
- application programming interface (API), 208–10
- arrays: LaTeX and, 236–37; R and, 260–61; scientific computing and, 186–97, 200–2
- arguments: LaTeX and, 224; Python and, 90, 113, 121–24, 133–36, 140–41, 167, 379; R and, 253, 269, 275–77, 280, 282–83, 286–87, 297; scientific computing and, 194, 204, 212; Unix and, 18–23, 25, 36, 44–47, 51
- assertions, 144–45
- AttributeError, 92, 137
- autocompletion, 21, 30, 90, 204n6
- automation, 2; basic programming and, 81; coding and, 135, 150, 366; data wrangling and, 300, 326; practical value of, 9, 368; scientific computing and, 193, 208; scientific typesetting and, 222–23, 234, 244–45; statistical computing and, 250, 267, 285; systematic errors and, 368; Unix and, 4, 13–14, 35, 43, 54; version control and, 55
- Axelrod, R., 218
- backups, 55, 359–61, 367
- bar plots, 292, 319, 323–25, 327–28
- Bash, 13–21, 44–46, 49–53, 136n3
- Basic Local Alignment Search Tool (BLAST), 174, 208, 212–13
- bibliographies, 221–22, 226, 242–44, 247
- BibTeX, 221–22, 242–44, 247
- binary formats, 61, 65, 82, 113, 337, 343, 364
- Biopython: handles and, 209–15; retrieving National Center for Biotechnology Information (NCBI) sequences and, 208–11; scientific computing and, 5, 185, 208–16, 219
- Bitbucket, 68, 78, 80, 370
- Bolstad, G. H., 335
- Booleans, 85, 88, 90, 193, 205, 343
- box plots, 207, 292–93, 320, 322–27, 330, 386
- branching: conditional, 105–7; merging and, 70–79; programming and, 105–7, 137; R and, 270–72; scientific typesetting and, 220; statistical computing and, 270–72; Unix and, 16; version control and, 55, 70–78
- breakpoint, 139, 144
- bugs: assertions and, 144–45; basic programming and, 82–83; correctness and, 3–4; debugging and, 4, 47, 67, 82–83, 120, 124, 127, 139–46, 181–82, 196, 283–84, 368; extemporaneous fixes and, 367–68; regular expressions and, 181–82; scientific computing and, 196; scientific typesetting and, 222; statistical computing and, 283–84; Unix and, 47; version control and, 66–67; writing good code and, 120, 124, 127, 139–49, 159
- built-in functions, 89–90, 92, 121, 249, 262, 275, 278, 281, 289
- Buzzard, V., 22–23, 25, 39–40, 53
- C, 82–83, 89, 105, 280, 369
- C++, 82, 166
- calculus, 197–99
- Chacon, S., 79
- chaining, 204
- Chang, Geoffrey, 144
- Chang, Winston, 326
- character-delimited files, 115–17
- chemical structures, 246
- chunks, 294–95
- CLI. *See* command-line interface (CLI)
- client-server systems, 339, 341
- clusters, 14, 323, 369
- coding: assertions and, 144–45; automation and, 135, 150, 366; branching and, 105–7, 137; bugs and, 4 (*see also* bugs); chunks and, 294–95; comma-separated values (CSV) format and, 161; comments and, 4, 17, 44, 72, 74–76, 84, 117, 135, 148, 181, 231–32, 251, 267, 342; copying objects and, 158–60; correctness and, 3–4, 155; data structures and, 128, 155–56, 158, 164; dictionaries and, 122–23, 129, 132, 137, 146, 151, 156–57, 159, 162; dividing task and, 367; doctest and, 147, 149–50, 152, 164;

- coding (cont.)
 - documentation and, 152, 154n7, 163–64, 294–95; encoding and, 113, 121, 191, 202, 337, 343; errors and, 120, 137–39 (*see also* errors); function scope and, 160–61; global scope and, 160–61; hacking and, 12, 368–69; hard coded files and, 44; importing and, 126–28, 131, 134, 136, 140, 144, 147, 150; indentation and, 105; Jupyter and, 131, 135, 139, 145, 147–48, 153–54, 163; keeping log and, 368; loops and, 155; modules and, 120–34, 136, 140–41, 147, 150, 152, 159, 164; mutable/immutable types and, 100–1, 155–58; NumPy module and, 139–40, 143; optimization and, 120, 153; organization and, 4, 120, 126–27, 133, 147; profiling and, 4, 82, 153–55; program structure and, 120–33; pseudocode and, 7, 9, 17, 117, 367–68; Python and, 4, 120–22, 126, 131–44, 147–60, 163–64; readability and, 2, 82, 86, 124, 133, 145, 154, 368; regular expressions and, 166 (*see also* regular expressions); repositories and, 154n7; science and, 4, 120–21, 153; SciPy module and, 128–30, 134, 144–45, 151–55; scripts and, 267–69; sets and, 132, 151; sharing and, 299, 370; simulations and, 127–28, 131–32, 135–36, 147, 153, 155; source code and, 82, 225, 228, 230, 238, 298; strings and, 121, 123, 131, 135–37, 140, 144, 147–50, 156–58; syntax and, 137 (*see also* syntax); terminals and, 136, 149; testing on paper and, 367; text editors and, 133, 135, 147, 154; tuples and, 128–29, 132, 156–57; unit testing and, 120, 146–52, 164; Unix and, 136; variables and, 122–23, 127, 135, 137, 139–44, 151, 155–56, 158, 160–61, 278; writing functions and, 121–26; writing style and, 133–35
- command-line interface (CLI), 12–15, 250
- comma-separated values (CSV) format: basic programming and, 112, 115–18; coding and, 161; data wrangling and, 301–2, 305, 313–14, 318, 320, 331, 335; modules and, 115–17; relational databases and, 338, 343–45, 361; scientific computing and, 190, 202, 208, 217; scientific typesetting and, 223, 235; SQLite and, 343–45, 361; statistical computing and, 264–65, 295, 297; Unix and, 25–41, 44–46, 53; version control and, 61, 66
- comments: coding and, 4, 17, 44, 72, 74–76, 84, 117, 135, 148, 181, 231–32, 251, 267, 342; LaTeX and, 231–32; regular expressions and, 181; relational databases and, 342; scientific typesetting and, 231–32; statistical computing and, 251, 267; Unix and, 17, 44; version control and, 72, 74–76
- commits: amending an incomplete, 66–68; checking out old, 76–77; deleting changes and, 68; Git and, 55–79, 374; repositories and, 55, 59, 61–63, 66–72, 77; reverting to last, 68; staging and, 61, 63, 67–69, 73–74, 77; version control and, 55–79, 374; writing messages associated with, 57
- compilation, 220, 232
- compilers, 82, 221, 226, 249
- Comprehensive R Archive Network (CRAN), 279, 293–94, 298
- Comprehensive TeX Archive Network, 246
- computing toolbox: automation and, 2, 13–14, 222, 234, 244; correctness and, 3–4, 155; documentation and, 2, 9, 83, 133, 146, 250, 267, 294–95; openness and, 3, 370; organization and, 1 (*see also* organization); pipes and, 1, 4–5, 366; readability and, 3 (*see also* readability); reproducibility and, 2, 13, 56, 80, 245; research potential and, 1; simplicity and, 3, 82, 128
- Comte, L., 355, 358, 363
- concatenate, Python and, 88, 91–92, 155, 205; R and, 255–256, 259, 271; Unix and, 25
- confidence intervals (CI), 289, 325–26
- coordinates, 100, 165, 183, 291, 319, 326, 353, 355
- copying objects, 19, 22–23, 33, 43, 50, 158–60
- correctness, 3–4, 155
- CRAN. *See* Comprehensive R Archive Network (CRAN)
- cross-references, 221, 226, 240
- CSV. *See* comma-separated values (CSV) format
- Cumming, G., 325n5
- cursors, 18, 84, 362–63
- Dalziel, B., 41, 115–17, 374, 376
- data dumps, 359–60
- data frames: data wrangling and, 302, 312; Python and, 202–6, 379; R and, 263–65, 280, 287, 295–97, 363; relational databases and, 342, 363
- data sets: basic programming and, 83, 117; complex, 1, 13, 185; computing toolbox and, 1–2; large, 1, 5, 13, 30, 30n7, 32, 66, 117, 185, 206–7, 249, 300, 337–39, 364; manipulation of, 300, 304–5, 309n2, 310, 314, 317, 323, 326; relational databases and, 337–39, 364; scientific computing and, 185, 202–3, 206–8; statistical computing and, 249, 263, 267, 270, 272, 280; Unix and, 13, 30n7, 32, 51; visualization and, 300, 304–5, 309n2, 310, 314, 317, 323, 326
- data structures: arithmetic of, 155–56; arrays and, 260; coding and, 126–28, 131, 134, 136, 140, 144, 147, 150, 155–56, 158, 164; copying objects and, 158–60; data wrangling and, 300–2; dictionaries and, 83, 93, 96–104, 108–9, 116–18, 122–23, 129, 132, 137, 146, 151, 156–57, 159, 162, 176, 209, 214; keys and, 93, 96–105, 108–9, 116, 118, 123, 137, 151, 157, 176, 209, 242, 340–41, 353–56, 359; lists and, 93–96, 110; matrices and, 52, 185–86, 196, 236–37, 257–61,

- 281; programming and, 4, 83, 88, 93–103; R and, 255–64; relational databases and, 337; scientific computing and, 185–86, 202, 209n7; sets and, 101–2; statistical computing and, 255–64; tuples and, 93, 100–4, 108–10, 128–29, 132, 156–57, 198, 379; vectors and, 185–86, 195–96, 200, 253, 255–58, 261–62, 272–73, 276–83, 380–81
- data types: Python and, 83, 85–89, 102, 117; regular expressions and, 176; relational databases and, 342–43, 345, 348; scientific computing and, 186–88; SQLite and, 342–43; statistical computing and, 251, 253–55, 262
- data wrangling: automation and, 300, 326; comma-separated values (CSV) format and, 301–2, 305, 313–14, 318, 320, 331, 335; data frames and, 302, 312, 384; data structures and, 300–2; documentation and, 301, 317–18, 320, 331, 333, 336; efficient data analysis and, 300; errors and, 304, 325–26; filters and, 305–6, 316, 333; gathering and, 313–15; grouped data and, 310–13, 333, 335, 352, 357; indexes and, 305; inner join and, 316–18, 356–58, 385, 387; libraries and, 301–2, 313, 320, 332; organization and, 5, 300, 313–15; outer join and, 316–17, 356–57; packages and, 5, 300–4, 307–10, 313, 318, 320, 331, 336; pipes and, 5, 307–9, 333; plotting and, 300, 318–26, 336; programming and, 82; R and, 300–1, 304, 307–8, 318–19, 326, 336; readability and, 300–1, 304, 308; regression analysis and, 265, 289–90, 319, 324, 335; renaming columns and, 308–9; sandboxes and, 302, 332; science and, 318; scientific computing and, 202; selecting data and, 304–10; sets and, 5, 300, 326; spreading and, 315–16; statistical computing and, 249, 263, 267, 270, 272, 280, 287–88, 291, 297, 310–13; strings and, 301–2, 324; tables and, 301, 313, 315–17; text files and, 302; tibbles and, 301–11, 314–18, 326, 333–34; tidyverse and, 5, 300–4, 308, 309n2, 313–18, 320, 336; tips/tricks for, 333–34; variables and, 304–10, 313–14, 319, 321–24, 327–29, 333; visualization and, 1, 5, 300, 318–33, 383–87
- debugging. *See* bugs
- default value, 98, 124, 140, 277–78, 286
- dictionaries: coding and, 122–23, 129, 132, 137, 146, 151, 156–57, 159, 162; data structures and, 83, 93, 96–104, 108–9, 116–18, 122–23, 129, 132, 137, 146, 151, 156–57, 159, 162, 176, 209, 214; keys and, 93, 96, 98–104, 108–9, 116, 118, 151, 157, 176, 209; Python and, 83, 93, 96, 132, 151, 209, 214; regular expressions and, 176; scientific computing and, 209, 214
- differential equations, 197–99
- directory, 15–17, 20, 59
- docstrings, 131, 135, 140, 147–50
- documentation, 2; basic programming and, 83–84, 90, 119; code and, 294–95; data wrangling and, 301, 317–18, 320, 331, 333, 336; Python and, 119; R and, 294–95; regular expressions and, 183n2; relational databases and, 352; scientific typesetting and, 239, 244–45; statistical computing and, 250, 262, 267, 278, 287, 294–98; Unix and, 50; version control and, 74, 78n5; writing good code and, 133, 146, 152, 154n7, 163–64
- dplyr, 301, 304, 307, 310, 312, 317–18, 350
- Dropbox, 56, 341
- dynamic typing, 89
- encoding, 113, 121, 191, 202, 337, 343
- Entrez Programming Utilities, 209–10, 214–15
- errors: AttributeError, 92, 137; automation and, 368; bars and, 325–26, 386; basic programming and, 88–89, 92–94, 100; bugs and, 3 (*see also* bugs); data wrangling and, 304, 325–26; exceptions and, 137–39; IndentationError, 137; IndexError, 94, 137; IOError, 137; KeyError, 137; NameError, 137; plotting experimental, 325–26; Python and, 137–39; relational databases and, 339, 352; scientific computing and, 187, 196; statistical computing and, 267, 283, 290; syntax, 137, 352; TypeError, 88, 100, 137, 156, 187; Unix and, 19, 49; writing good code and, 120, 137–45, 148, 153, 156
- Excel, 202, 366
- Extensible Markup Language (XML), 166, 208–9, 212–13, 222, 301
- faceting, 319, 328–29
- FASTA files, 51, 176, 210–11, 245, 273
- Fauchald, P., 302, 305, 314, 320, 331, 384–85
- Fermat's last theorem, 7
- filters: data wrangling and, 305–6, 316, 333; scientific computing and, 205; SQLite and, 348–50, 352, 365; visualization and, 323–31
- flipped classroom, 9
- floats, 85, 88, 90, 97, 103, 157
- folder. *See* directory
- forks, 78–79
- FORTRAN, 82, 89
- Fox, C. W., 217
- free software, 3, 56, 222, 249–50, 341
- function scope, 160–61
- Gächter, S., 207–8
- game theory, 218
- geometry, 319–23, 333
- Gesquiere, L. R., 51
- ggplot, 249, 320–32, 385–87
- Git, 9; commits and, 55–79, 374; configuring, 57; deleting changes and, 68; forks and, 78–79; getting help in, 58; getting started with, 56–58;

- Git (cont.)
 - ignoring files/directories and, 65–66; indexes and, 61, 64, 66; installing, 57; moving/removing files and, 66; pull request and, 79; repositories and, 56–59, 61, 64–72, 77–78, 80; showing changes and, 64–65; stashing and, 69–70; troubleshooting, 66–68; unstaging files and, 67; version control and, 4, 56–73, 77–80; workflow and, 56, 58–64, 66, 69, 79
- Git Bash 13–21, 50, 136n3
- GitHub, 68–69, 78–80, 359, 368, 370
- global scope, 160–61
- GNU Scientific Library, 82
- Goldberg, E. E., 295
- grammar of graphics, 319–20
- grants, 221, 246
- graphical user interface (GUI), 13, 243, 362
- graphix, 237–39
- greedy quantifiers, 171, 175
- grouped data, 310–13, 333, 335, 350–52, 357
- groups, 179–80, 350–52
- guide, 330–31, 336
- hacking, 12, 368–69
- Hamilton, W. D., 218n20
- handles, 112, 209–15
- hard coded files, 44
- heuristics, 370
- histograms, 207, 291–93, 320–22, 327, 333, 387
- history: R and, 251; relational databases and, 343; scientific computing and, 214, 216, 218; scientific typesetting and, 222; Unix and, 14, 18, 30, 50; version control and, 4, 55–56, 62–63, 66–68, 78
- HTML files, 166, 220, 222, 294
- IDE. *See* integrated development environment (IDE)
- immutable types, 100–1, 155–58
- importing: coding and, 126–28; libraries and, 278–79; modules and, 115, 126–28, 131, 134, 136, 140, 147, 150; packages and, 115, 126–28, 131, 134, 136, 140, 147, 150, 202; tables and, 343–44
- indentation, 105, 122, 133, 137, 181
- IndexError, 94, 137
- indexes: data wrangling and, 305; LaTeX and, 226; Python and, 93–96, 100, 110, 137; R and, 255, 259, 261, 266; regular expressions and, 176; relational databases and, 338–39, 353–54; scientific computing and, 203–4; scientific typesetting and, 226; Shannon’s diversity, 365; statistical computing and, 255, 259, 261, 266; version control and, 61, 64, 66
- inner join, 316–18, 356–58, 385, 387
- integrated development environment (IDE), 369; Jupyter and, 10, 83–85, 105, 107, 117, 131, 135, 139, 145–48, 153–54, 163, 166, 176, 202, 207, 211, 221; RStudio and, 10, 250–54, 269–73, 279, 284–85, 293n10, 294, 301–2, 304, 308
- integration, 197–99
- interquartile range (IQR), 322
- Introduction to Scientific Computing, 8
- IOError, 137
- ipdb, 142–43
- Java, 82, 166, 369
- Jiang, Y., 161
- Jonker, L. B., 199
- JSON files, 166
- Jupyter: conditional branching and, 105; exporting code and, 84, 131, 135; installing, 83–84; integrated development environment (IDE) and, 10, 83–85, 105, 107, 117, 131, 135, 139, 145–48, 153–54, 163, 166, 176, 202, 207, 211, 221; looping and, 107; programming and, 83–85, 105, 107, 117; regular expressions and, 166, 176; scientific computing and, 202, 207, 211; scientific typesetting and, 221; writing good code and, 131, 135, 139, 145, 147–48, 153–54, 163
- Kacsoh, B. Z., 191
- Kendall’s correlation, 297, 318, 385
- kernels, 12, 57, 107
- KeyError, 137
- keys: data structures and, 93, 96–105, 108–9, 116, 118, 123, 137, 151, 157, 176, 209, 242, 340–41, 353–56, 359; dictionaries and, 93, 96, 98–104, 108–9, 116, 118, 151, 157, 176, 209; foreign, 340, 341, 353–55, 359; primary, 340, 353–55
- Kirsch, Daniel, 231n5
- Knauff, M., 248
- Knuth, Donald, 153, 222
- labels, 379–82; data wrangling and, 2318, 324, 329–30; scientific computing and, 199–200, 203–5; scientific typesetting and, 240, 245; statistical computing and, 261, 291–92
- Lahti, L., 162
- Lampert, Leslie, 222
- LaTeX: alignment and, 232; arrays and, 236–37; bibliographies and, 221–22, 226, 242–44, 247; BibTeX and, 221–22, 242–44, 247; chemical structures and, 246; commands and, 228–29; comments and, 231–32; compilation and, 220–21, 226, 232; Comprehensive TeX Archive Network and, 246; cross-references and, 240; directory structure and, 16; document classes and, 224; document structures and, 223–27; environments and, 228–29; figures and, 237–39; font size and, 224; indexes and, 226; installing, 223; justification and, 232; labels and, 240; lists and, 240–41; long documents and, 232–33;

- Markdown and, 220–21, 294; math and, 221, 229–31; matrices and, 236–37; new lines and, 228; packages and, 220, 222–25, 229, 234–35, 238, 244, 294; reasons for using, 220–23; scientific typesetting and, 5–6, 220–48; sequence alignments and, 245; space and, 228; special characters and, 228; stability of, 222; statistical computing and, 294; steep learning curve of, 220–21, 223; syntax and, 11; tables and, 233–36; templates and, 222, 224, 244, 246–47; typesetting text with, 228–44; version control and, 61; Visual FAQ for, 247
- legends, 200, 327, 330–31, 379
- Letchford, A., 297
- libraries, 82; data wrangling and, 301–2, 313, 320, 332; Python and, 363; R and, 278–79; relational databases and, 5, 363; scientific computing and, 5, 185, 187, 191, 202, 207–8, 216, 219; statistical computing and, 278–79
- line terminators, 26, 43n11, 50
- Linux, 10, 12, 14, 57
- list comprehension, 110
- lists: data structures and, 93–96, 110; itemized, 241; LaTeX and, 240–41; numbered, 241; R and, 261–62
- literal characters, 168, 175
- loops: coding and, 155; Jupyter and, 107; modifying behavior of, 107–9; programming and, 107–12, 114, 137, 376; Python and, 4, 109, 111, 137; R and, 272–74; regular expressions and, 178; scientific computing and, 213, 216; statistical computing and, 272–77, 280–81, 297; Unix and, 47–49, 52n17
- Malhotra, A., 245
- mapping, 319, 321, 326–27, 332, 386
- Markdown, 84, 220–21, 294–95, 319
- markup languages, 84n2, 166, 208–9, 212–13, 220, 222, 223, 294
- Martin, T. E., 335
- match object, 167, 178
- MATLAB, 82, 249, 298
- Matplotlib, 183, 191, 199–200, 207, 216, 249, 379
- matrices, 52, 185–86, 196, 236–37, 257–61, 281
- maximum: data wrangling and, 311, 334; relational databases and, 370; scientific computing and, 194, 209n9; writing good code and, 141
- Maynard Smith, J., 217
- mean: basic programming and, 118; data wrangling and, 310–13, 325–26; R and, 256, 259, 269, 279, 281, 288–89, 291; scientific computing and, 187, 192, 195, 206; writing good code and, 141, 151–52, 161–62
- median, 206, 256, 288, 290, 310, 322
- merging, 70–79
- metacharacters, 34, 168–74
- metadata, 55, 162, 223, 367
- metaheuristics, 370
- Microsoft: Excel, 202, 366; Office, 341; Windows, 10, 13–15, 17, 21, 26, 43, 50, 136n3, 176n1, 222; 43; Word, 220, 248
- Mikaelyan, A., 364–65
- Miller, G., 144
- minimum, 53, 141, 187, 311
- modulo, 86, 252
- mutable types, 100–1, 155–58
- NameError, 137
- namespace pollution, 127
- National Center for Biotechnology Information (NCBI), 172–73, 208–14
- Nejasmic, J., 248
- next generation sequencing (NGS) data, 51
- nimble computer language, 280
- nonprinting characters, 50
- normalization, 313, 353
- normalized difference vegetation index (NDVI), 305–10, 313, 317, 320–23, 334, 383
- NumPy module: arrays and, 186–90; coding and, 139–40, 143; image processing with, 191–93; random distributions and, 194–96; randomization and, 194–96; scientific computing and, 185–203, 216, 219, 377–78; statistical computing and, 249; uniform distribution and, 194–96
- object oriented programming, 90
- OpenBSD, 12
- openness, 3
- Open Science movement, 370–71
- operators: data wrangling and, 308; LaTeX and, 230–31; Python and, 85–86, 94, 101, 134, 155; R and, 252–53, 269; scientific computing and, 186, 205; Unix and, 33
- optimization, 120, 153, 200–2, 280, 370
- organization: automation and, 2; coding and, 4, 120, 126–27, 133, 147; data wrangling and, 5, 300, 313–15; efficiency and, 1; programming and, 81, 93, 116; relational databases and, 337, 340, 352–54; reproducibility and, 2; scientific computing and, 202, 217; scientific typesetting and, 222, 242; statistical computing and, 255, 263, 267, 277; Unix and, 15; version control and, 4, 55–56
- OS X, 10; regular expressions and, 176n1; scientific typesetting and, 222; Unix and, 12–19, 43, 50n14
- outer join, 316–17, 356–57
- Pacifici, M., 30–32, 34–35, 39–40, 44–46, 373
- pandas, 379; chaining and, 204; relational databases and, 363; scientific computing and,

- pandas (cont.)
 - 185, 202–8, 216–17, 219; statistical computing and, 249
- parallel computing, 369
- parallel tempering, 370
- parsers, 168, 172, 209–14, 287n9
- PDF files, 61, 84, 220, 222, 225–26, 231n5, 243, 294, 319
- Perl, 82, 166
- pickled objects, 132–33, 146
- pipes, 81; alternations and, 172; computing toolbox and, 1, 4–5, 366; data wrangling and, 5, 307–9, 333; Python and, 135; R and, 307–9, 333; redirection and, 27–29; regular expressions and, 172; Unix and, 13, 27–35, 38–39, 43, 46
- plotting: 3D, 293; bar, 292, 319, 323–28; box, 207, 292–93, 320, 322–27, 330, 386; confidence intervals (CI) and, 289, 325–26; coordinates and, 100, 165, 183, 291, 319, 326, 353, 355; experimental errors and, 325–26; faceting and, 328–30; frequency distribution and, 321–22; ggplot, 249, 320–32, 385–87; ggplot2, 5, 216, 290, 300–1, 318–24, 327–28, 330, 335–36; histogram, 207, 291–93, 320–22, 327, 333, 387; interquartile range (IQR) and, 322; legends and, 200, 327, 330–31, 379; Python and, 191, 216; R and, 268, 290–93; saving and, 332–33; scales and, 326–28; scatter, 183, 207, 291, 319, 324–26, 333, 385; setting a feature and, 332; standard deviation and, 279, 312–13, 325–26; standard error of the mean (SEM) and, 325; structure and, 320–21; themes and, 331–32; violin, 320–23, 325n4; visualization and, 300, 318–26, 333
- primers, 165, 179–80, 208
- prisoner's dilemma, 218
- profiling, 4, 82, 153–55
- programming: application programming interface (API) and, 208–10; automation and, 81; basic, 4, 81–119; Booleans and, 85, 88, 90, 193, 205, 343; branching and, 105–7, 137; bugs and, 82–83 (*see also* bugs); choosing a language, 81–83; comma-separated values (CSV) format and, 112, 115–18; common functions and, 103–5; compilers and, 82, 221, 226, 249; data sets and, 83, 117 (*see also* data sets); data structures and, 4, 83, 88, 93–103; documentation and, 83–84, 90, 119; dynamic typing and, 89; errors and, 88–89, 92–94, 100 (*see also* errors); flow and, 105–12; hacking and, 12, 368–69; Jupyter and, 83–85, 105, 107, 117; loops and, 107–12, 114, 137; modules and, 88, 92, 100, 115–17; organization and, 81, 93, 116; profiling and, 82; Python and, 6, 82–93, 96, 100, 105–19; R and, 82; readability and, 2, 82, 86, 124, 133, 145, 154; reasons for doing, 81; science and, 81–82, 185; scripting and, 43–47, 362; simplicity and, 82, 128; simulations and, 82–83; source code and, 82, 225, 228, 230, 238, 298; strings and, 83, 85, 88–92, 94, 97, 101, 103–4, 108–10, 113; syntax and, 83, 84n2 (*see also* syntax); terminals and, 84, 107; text editors and, 105; text files and, 112–15; time series and, 117–18; tuples and, 93, 100–4, 108–10; unit testing and, 82; working with files and, 112–17
- program structure, 120–33
- pseudocode, 7, 9, 17, 117, 367–68
- PubMed, 183, 214–17
- pull request, 79
- p-value, 161–62, 289–90, 297, 382
- Python: Anaconda and, 84, 176n1; application programming interface (API) and, 208–10; arguments and, 90, 113, 121–24, 133–36, 140–41, 167, 379; assignment and, 83, 87–89, 100, 137, 158, 160n8; basic programming and, 82–93, 96, 100, 105, 107–15; Biopython and, 5, 185, 208–16, 219; built-in functions of, 89–90; coding and, 4, 120, 122, 126, 131–44, 147–60, 163–64; command line and, 135–36; comments and, 84, 117, 120, 135, 148, 181 concatenate and, 88, 91–92, 155; copying objects and, 158–60; data frames and, 202–6, 379; data structures and, 83, 93, 155–56, 164, 202, 208; data types and, 83, 85–89, 102, 117; data wrangling and, 384; dictionaries and, 83, 93, 96, 132, 151, 209, 214; directory structures and, 16, 21–22; docstrings and, 131, 135, 140, 147–50; documentation and, 119; dynamic typing and, 89; errors and, 137–39; exceptions and, 137–39; exponentiation and, 86; floats and, 85, 88, 90, 97, 103, 157; getting help in, 84–85; handles and, 209–15; indentation and, 105; indexes and, 93–96, 100, 110; installing, 83–84; libraries and, 363; loops and, 4, 109, 111, 137; match object and, 167, 178; methods and, 90, 112, 191, 211; modules and, 5, 88, 92, 100, 115, 120, 126, 131–32, 134, 136, 141, 147, 150, 152, 164, 166–67, 176, 183n2, 185, 187, 212, 216, 369; mutable/immutable types and, 155–58; object oriented programming and, 90; operators and, 85–86, 94, 101, 134, 155; packages and, 84, 120, 126, 166, 185, 207, 216, 219, 249, 300, 341, 369; pandas and, 185, 202–8, 216–17, 219, 249, 363, 379; performance and, 369; permissions and, 136n4; pickled objects and, 132–33, 146; pipes and, 135; plotting and, 191, 216; practicing, 366; profiling and, 4, 82, 153–55; programming and, 6, 83–93, 96, 100, 105–19; queries and, 89, 208, 362; regular expressions and, 5, 166–67, 173, 176n1, 177–78, 183–84; relational databases and, 341, 362–65; scientific computing and, 5, 185, 187, 191, 193, 198, 202, 207–16, 219; scientific typesetting and, 225n3; sets and, 83, 93, 132, 169–70, 185, 202, 207, 249, 300; simple

- calculations in, 85–87; slicing and, 94, 100, 193, 211; SQLite and, 341, 362–63, 365; statistical computing and, 249, 255, 262, 278, 299–300; strings and, 83, 85, 88–92, 113, 131, 135, 137, 147, 149, 173; syntax and, 11, 49; variables and, 82–83, 87–91, 96, 102, 135, 137, 144, 155–61, 249, 278; Windows and, 176n1; writing good code and, 120–22, 126, 131–44, 147–60, 163
- quantifiers, 170–72, 175
- queries: Python and, 89, 208, 362; relational databases and, 337, 342–52, 356, 358, 360–63; scientific computing and, 183, 208–10, 213–16; SQLite and, 5, 341–52, 355–65 (*see also* Structured Query Language (SQL)); statistical computing and, 294
- R, 10, 369; arguments and, 253, 269, 275–77, 280, 282–83, 286–87, 297; arrays and, 260–61; assignment and, 251, 253–55, 269; basic plotting and, 290–93; basic programming and, 82; branching and, 270–72; Chang and, 326; chunks and, 294–95; command-line interface of, 250, 285–87; Comprehensive R Archive Network (CRAN) and, 279, 293–94, 298; concatenate and, 255–256, 259, 271; data frames and, 263–65, 280, 287, 295–97, 363; data structures and, 255–64; data types and, 251, 253–55, 262; data wrangling and, 300–1, 304, 307–8, 318–19, 326, 336; debugging, 283–84; documentation and, 294–95; finding help in, 251; flow of, 270–74; as free software, 250; functions and, 275–78; getting started with, 251–53; importing libraries and, 278–79; indexes and, 255, 259, 261, 266; installing, 250; interfacing with operating system and, 284–85; libraries and, 278–79; lists and, 261–62; logical operators of, 251–53; loops and, 272–74; Markdown and, 221, 294–95, 319; matrices and, 257–61; operators and, 252–53, 269; performance and, 369; plotting and, 268, 290–93; reading data and, 264–66, 301–2; reasons for using, 249–51; relational databases and, 342, 345, 362–75; renaming columns and, 308–9; scientific computing and, 202, 207, 216; scientific typesetting and, 221–23, 225n3, 234; scripts and, 267–69; selecting data and, 304–10; sequences and, 257, 273; simplicity and, 82; SQLite and, 363–64; statistical computing and, 5–6, 249–99, 366; statistics in, 287–90; strings and, 262; syntax and, 11; tidyverse and, 5, 300–4, 308, 309n2, 313, 318, 320, 336; variables and, 249, 253–54, 257, 262–63, 270, 272–73, 277–78, 280, 284, 286–90, 309–10; vectors and, 253, 255–58, 261–62, 272–73, 276–83; visualization and, 82; Wickham and, 300
- random numbers, 128, 152, 194–96, 203, 277, 279–80
- Rapoport, Anatol, 218n20
- RDBMS. *See* relational database management system (RDBMS)
- readability: binary files and, 61; cleverness and, 3; coding and, 2, 82, 86, 124, 133, 145, 154, 368; data wrangling and, 300–1, 304, 308; programming and, 2, 82, 86, 124, 133, 145, 154; regular expressions and, 181; reproducibility and, 2; scientific computing and, 208; simplicity and, 3; statistical computing and, 267, 296; Unix and, 21, 46; version control and, 61
- recursive function, 22–24, 39, 42, 75, 126
- redirection, 27–29, 33, 362
- redundancy, 338, 341, 352–54
- regression analysis, 265, 289–90, 319, 324, 335
- regular expressions: alternations and, 172; anchors and, 171–72; bugs and, 181–82; building, 167–75; comments and, 181; data types and, 176; dictionaries and, 176; documentation and, 183n2; greedy quantifiers and, 171, 175; groups in, 179–80; Jupyter and, 166, 176; literal characters and, 168, 175; loops and, 178; match object and, 167, 178; metacharacters and, 34, 168–74; modules and, 166–67, 175–79, 183n2; packages and, 166, 182; pipes and, 172; Python and, 5, 166–67, 173, 176n1, 177–78, 183–84; quantifiers and, 170–72, 175; queries and, 337, 342–52, 356, 358, 360–63; quest for perfect, 181–82; readability and, 181; reasons for using, 165–66; re module and, 175–79; science and, 182–83; sets and, 169–70; shells and, 166; to shorten code, 166; sophisticated replace of, 166; strings and, 165–75, 177, 179, 181; syntax and, 165–67; terminals and, 176n1; text editors and, 166–67; text files and, 165–66; Unix and, 165; variables and, 179–80, 214; verbose, 181; visualization and, 183
- relational database management system (RDBMS), 337, 341–43
- relational databases: accessing, 362–63; backups and, 359–61; client-server systems and, 339, 341; comma-separated values (CSV) format and, 338, 343–45, 361; comments and, 342; connecting to, 362–63; data frames and, 342, 363; data sets and, 337–39, 364; data structures and, 337; data types and, 342–43, 345, 348; designing, 352–55; disadvantages of, 340; documentation and, 352; dumping data and, 359–60; errors and, 339, 352; grouped data and, 352, 357; graphical user interfaces (GUIs) and, 362; indexes and, 176, 338–39, 353–54; inner join and, 316–18, 356–58, 385, 387; integrity and, 339; libraries and, 5, 363; organization and, 337, 340, 352–54; outer join and, 316–17, 356–57; packages and, 341, 363, 365; pandas

- relational databases (cont.)
 - and, 363; Python and, 341, 362–65; R and, 342, 345, 362–75; reasons for using, 338–40; record handling and, 360–61; restoring, 359–60;
 - scripting and, 362; security and, 339; sets and, 337–39, 364; SQLite and, 5, 341–52, storage and, 338; strings and, 343; structure of, 340–41;
 - syntax and, 348, 352, 356, 361; tables and, 337–38, 340–47, 352–61, 364–65; terminals and, 341–44, 356, 359, 361–62; text editors and, 340; text files and, 337–40, 359, 362; Unix and, 345, 350; views and, 358–59, 361–62
- relative path, 21–23, 40
- re module, 5, 126, 166–67, 175–79
- repositories: coding and, 154n7; commits and, 55, 59, 61–63, 66–72, 77; contributing to public, 78–79; forks and, 78–79; Git and, 56–59, 61, 64–72, 77–78, 80; local, 56, 58, 68–69, 78; pull request and, 79; remote, 68–70; scientific computing and, 208; stashing and, 69–70; time stamps and, 370–71; version control and, 55–73, 77–80
- reproducibility, 245; documentation and, 2, 294–95; organization and, 2; readability and, 2; science and, 2; simulations and, 2; Unix and, 13; version control and, 56, 80
- RGB images, 191–92
- root directory, 15, 16, 20, 59
- Rstudio: as integrated development environment (IDE), 251; reasons for using, 250–51; statistical computing and, 10, 250–54, 269–73, 279, 284–85, 293n10, 294, 301–4, 308
- RTF files, 222
- Ruby, 82, 166
- Ruggiero, Michael, 182

- Saavedra, S., 29, 39–41, 52
- sampling, 161, 313, 338, 353–54
- sandboxes: basic programming and, 115; coding and, 131, 135, 145, 147; data wrangling and, 302, 332; regular expressions and, 176; relational databases and, 341, 343–44, 355; scientific computing and, 191, 202; scientific typesetting and, 225; statistical computing and, 265, 269–70, 297; Unix and, 16, 20, 22, 27–29, 34, 39–42, 51; version control and, 58, 70–71, 78
- Sander, Liz, 370n3
- scales, 208, 239–40, 312–13, 318–19, 326–29
- scatter plots, 183, 207, 290–91, 319, 324–26, 333, 385
- Schulz, J. F., 207–8
- science: automation and, 2; basic programming and, 81–82; coding and, 4; correctness and, 3; data wrangling and, 318; map of, 182–83; Open, 3, 370–71; programming and, 185; regular expressions and, 182–83; reproducibility and, 2; scientific computing and, 185, 191, 219; scientific typesetting and, 221–22, 242; as software development, 4; statistical computing and, 297; version control and, 4, 78–79; writing good code and, 120–21, 153
- scientific computing: application programming interface (API) and, 208–10; arrays and, 186–97, 200–2; automation and, 193, 208; BLAST and, 208, 212–13; bugs and, 196; chaining and, 204; comma-separated values (CSV) format and, 190, 202, 208, 217; data sets and, 185, 202–3, 206–8; data structures and, 185–86, 202, 209n7; data types and, 186–88; data wrangling and, 202; dictionaries and, 209, 214; errors and, 187, 196; filters and, 205; handling and, 209–15; indexes and, 203–4; Jupyter and, 202, 207, 211; libraries and, 5, 185, 187, 191, 202, 207–8, 216, 219, 278–79; linear algebra and, 185, 196–97; loops and, 213, 216; modules and, 185, 187, 209n7, 210–12, 216; NumPy module and, 185–203, 216, 219, 377–78; organization and, 202, 217; packages and, 5, 185–86, 196–97, 200, 202, 207, 209, 216, 219; pandas and, 185, 202–8, 216–17, 219; Python and, 5, 185, 187, 191, 193, 197–98, 202, 207–16, 219; queries and, 183, 208–10, 213–16; R and, 202, 207, 216, 249–99; readability and, 208; repositories and, 208; science and, 185, 191, 219; SciPy module and, 185, 196–202, 216, 219, 377–79; SeqIO module and, 210–11; sets and, 185, 202, 206–7; simulations and, 5, 185, 194–96; tuples and, 198; vectors and, 185–86, 195–96, 200, 253, 255–58, 261–62, 272–73, 276–83, 380–81; visualization and, 185, 191, 208; workflow and, 208
- scientific typesetting, 5; arrays and, 236–37; automation and, 222–23, 234, 244–45; bibliographies and, 221–22, 226, 241–44, 247; branching and, 220; bugs and, 222; commands and, 228–29; comma-separated values (CSV) format and, 223, 235; comments and, 231–32; cross-references and, 221, 226, 240; documentation and, 239, 244–45; environments and, 228–29; figures and, 237–39; fonts and, 224, 241–42; indexes and, 226; Jupyter and, 221; labels and, 240; LaTeX and, 220–48; lists and, 240; long documents and, 232–33; math and, 229–31; matrices and, 236–37; organization and, 222, 242; OS X and, 222; packages and, 220–25, 229, 232–39, 244–46; Python and, 225n3; R and, 221–23, 225n3, 234; reasons for using, 249; reproducibility and, 245; sandboxes and, 225; science and, 221–22, 242; special characters and, 50, 85, 105, 173, 228; syntax and, 221–22; tables and, 233–36, 264–65, 288, 301, 313, 315–17, 337–38, 340–47, 352–61, 364–65; templates and, 222, 224, 244, 246–47; terminals and, 226, 243; text editors and, 220,

- 222, 225, 228; text files and, 220–21, 245;
- Ubuntu and, 222; Windows and, 222
- SciPy module: bibliography and, 242–44; calculus and, 197–99; coding and, 128–30, 134, 144–45, 151–55; font styles and, 241–42; linear algebra and, 185, 196–97; optimization and, 200–2; packages and, 244–46; scientific computing and, 185, 196–202, 216, 219, 377–79; statistical computing and, 249
- scope, 160–61, 278
- scripts, 43–47, 267–69, 362
- security, 41, 70n4, 339
- sequence alignments, 213, 244–45
- sets: automation and, 2; coding and, 132, 151; complexity and, 1, 13, 185; data wrangling and, 5, 300, 326; initializing, 101–2; programming and, 83, 93, 101–2, 104, 117; Python and, 83, 93, 132, 169–70, 185, 202, 207, 249, 300; regular expressions and, 169–70; relational databases and, 337–39, 364; scientific computing and, 185, 202, 206–7; statistical computing and, 249, 270, 272; Unix and, 13, 30n7, 32
- Shannon's index of diversity, 365
- shells: command-line interface (CLI) and, 12–15, 250; getting started with, 17–22; ipdb, 142–43; regular expressions and, 166; statistical computing and, 285; Unix and, 12–22, 29, 43–44, 49, 54; version control and, 70n4, 71
- simulations: annealing and, 370; coding and, 127–28, 131–32, 135–36, 147, 153, 155; metaheuristics and, 370; programming and, 82–83; reproducibility and, 2; scientific computing and, 5, 185, 194–96; statistical computing and, 272, 279, 298; version control and, 61
- Smith, F. A., 296
- smoothing, 319, 324, 333
- source code, 82, 225, 228, 230, 238, 298
- special characters, 50, 85, 105, 173, 228
- spreading, 315–16
- spreadsheets, 35, 115, 202, 263, 304, 337–38, 353, 366–67
- SQL. *See* Structured Query Language (SQL)
- SQLite: basic queries and, 344–52; comments and, 342–43; CSV mode and, 343–45, 361; data types and, 342–43; dumping data and, 359–60; filters and, 348–50, 352, 365; group operations and, 350–52; installation of, 341; Python and, 341, 362–63, 365; queries and, 5, 341–52, 355–65; R and, 363–64; record handling and, 360–61; regular expressions and, 349; relational databases and, 5, 341–52, 355–65; running RDBMS of, 341–42; subsetting data and, 345–48; tables and, 343–45, 356–62; views and, 358–59, 361
- square root, 143, 188, 231, 252
- staging, 61, 63, 67–69, 73–74, 77
- standard deviation (SD), 279, 312–13, 325–26
- standard error of the mean (SEM), 325
- stashing, 69–70
- statistical computing: automation and, 250, 267, 285; basic plotting and, 290–93; branching and, 270–72; bugs and, 283–84; chunks and, 294–95; command-line interface (CLI) and, 250; comma-separated values (CSV) format and, 264–65, 295, 297; comments and, 251, 267; Comprehensive R Archive Network (CRAN) and, 293–94; confidence intervals (CI) and, 289, 325–26; counting and, 310–13; data sets and, 249, 263, 267, 270, 272, 280, 287–88, 291, 297; data structures and, 255–64; data types and, 251, 253–55, 262; data wrangling and, 310–13; documentation and, 250, 262, 267, 278, 287, 294–98; errors and, 267, 283, 290; grouped data and, 310–13; indexes and, 255, 259, 261, 266; LaTeX and, 294; loops and, 272–74, 276–77, 280–81, 297; NumPy module and, 249; organization and, 255, 263, 267, 277; packages and, 5, 250, 269, 278–79, 287, 290, 293–94, 298; pandas and, 249; Python and, 249, 255, 262, 278, 299–300; queries and, 294; R and, 5 (*see also* R); readability and, 267, 296; science and, 297; SciPy module and, 249; scripts and, 267–69; sets and, 249, 270, 272; shells and, 285; simulations and, 272, 279, 298; strings and, 254–55, 261–62, 265, 270, 278, 286; syntax and, 251, 295; tables and, 264–65, 288; terminals and, 250, 294; text files and, 250, 264, 267; Unix and, 285; vectors and, 253, 255–58, 261–62, 272–73, 276–83; workflow and, 285
- Stouffer, D. B., 52
- Straub, B., 79
- strings, 377; coding and, 121, 123, 131, 135–37, 140, 144, 147–50, 156–58; data wrangling and, 301–2, 324; programming and, 83, 85, 88–92, 94, 97, 101, 103–4, 108–10, 113; Python and, 83, 85, 88–92, 113, 131, 135, 137, 147, 149, 173; R and, 262; raw notation and, 173–75; regular expressions and, 165–75, 177, 179, 181; relational databases and, 343; retractions and, 144; statistical computing and, 254–55, 261–62, 265, 270, 278, 286; Unix and, 13, 15, 19, 27–28, 32, 38
- Structured Query Language (SQL): directory structure and, 16; indexing and, 338–39; RDBMS and, 337, 341–43; redundancy and, 338, 341, 352–54; relational databases and, 5, 337, 341–52, 355–65; SQLite and, 5, 341–52, 355–65; syntax and, 348, 352, 356, 361; wildcards and, 349–50
- sum, 104, 187, 190, 193, 203, 229–31, 256, 259, 334, 336
- syntax: Bash and, 49; basic programming and, 83, 84n2; editor choice and, 11; errors and, 137,

- syntax (cont.)
 - 352; LaTeX and, 11; Python and, 11, 49; R and, 11; regular expressions and, 165–67; relational databases and, 348, 352, 356, 361; scientific typesetting and, 221–22; statistical computing and, 251, 295; Unix and, 49; writing good code and, 137
- tables: data wrangling and, 301, 313, 315–17; exporting, 361–62; individual, 354; joining, 355–58; LaTeX and, 233–36; relational databases and, 337–38, 340–47, 352–61, 364–65; sampling, 353; scientific typesetting and, 233–36, 264–65, 288, 301, 313, 315–17, 337–38, 340–47, 352–61, 364–65; site, 353; species, 354; SQLite and, 343–44; statistical computing and, 264–65, 288
- Taylor, P. D., 199
- templates, 222, 224, 244, 246–47
- terminals: basic programming and, 84, 107; coding and, 136, 149; command-line interface (CLI) and, 12–15, 250; regular expressions and, 176n1; relational databases and, 341–44, 356, 359, 361–62; scientific typesetting and, 226, 243; statistical computing and, 250, 294; Unix and, 12–18, 43, 45n12, 49; version control and, 57–59, 62
- text editors, 10–11; basic programming and, 105; LaTeX and, 220, 222, 225, 228; line terminators and, 26, 43n11, 50; mastering, 369; regular expressions and, 166–67; relational databases and, 340; scientific typesetting and, 220, 222, 225, 228; Unix and, 24, 32, 43, 50; version control and, 57, 60–62, 65, 77; writing good code and, 133, 135, 147, 154
- text files: backups and, 367; basic programming and, 112–15; data wrangling and, 302; indexing and, 93, 100, 176, 204, 338–39; regular expressions and, 165–66; relational databases and, 337–40, 359, 362; scientific typesetting and, 220–21, 245; statistical computing and, 250, 264, 267; Unix and, 13–14, 24–27, 30, 32, 36, 43, 50; version control and, 61
- tibbles, 301–11, 314–18, 326, 333–34
- tidyverse: data wrangling and, 5, 300–4, 308, 309n2, 313–18, 320, 336; pipelines and, 307–9, 333; selecting data and, 304–10; tibbles and, 301–11, 314–18, 326, 333–34
- time series, 117–18
- time stamps, 370–71
- Torvalds, Linus, 12, 57
- tuples: coding and, 128–29, 132, 156–57; data structures and, 93, 100–4, 108–10, 128–29, 132, 156–57, 198, 379; immutable, 100–1, 157; programming and, 93, 100–4, 108–10; scientific computing and, 198
- TypeError, 88, 100, 137, 156, 187
- Ubuntu, 10; Linux and, 12; regular expressions and, 176n1; scientific typesetting and, 222; Unix and, 12–19, 43, 50
- unit testing: coding and, 120, 146–52, 164; doctest and, 147, 149–50, 152, 164; executing, 149–50; handling complex, 150–52; programming and, 82; random value and, 151–52; unordered object and, 151; writing the tests and, 147–49
- Unix, 304, 366; absolute path and, 21–22, 40; advanced commands and, 27–43; arguments and, 18–23, 25, 36, 44–47, 51; autocompletion and, 21, 30; automation and, 4, 13–14, 35, 43, 54; basic commands of, 18–19, 22–27; branching and, 16; bugs and, 47; character substitution and, 32–35; coding and, 136; column selection and, 29–32; command-line interface (CLI) and, 12–15; comma-separated values (CSV) format and, 25–41, 44–46, 53; comments and, 17, 44–47; computing toolbox and, 4, 6; concatenating files and, 25; copying and, 19, 22–23, 33, 43, 50 creating files and, 13, 22–23, 27–29, 33–36, 42–43, 46–51; data sets and, 30n7, 32, 51; directory structure of, 15–17, 20–22; documentation and, 50; errors and, 19, 49; file system of, 12; finding files and, 39–41; getting help in, 19–20; handling directories/files and, 22–24; hard coded files and, 44; history and, 14, 18, 30, 50; installation of, 14–15; line selection and, 36–39; line terminators and, 26, 43n11, 50; Linux and, 10, 12, 14, 57; loops and, 47–49, 52n17; moving files and, 23; next generation sequencing (NGS) data and, 51; nonprinting characters and, 50; OpenBSD and, 12; operators and, 33; organization and, 15; OS X and, 12–19, 43, 50n14; packages and, 50; permissions and, 41–43, 45; pipes and, 13, 27–35, 38–39, 43, 46, 308; readability and, 21, 46; reasons for using, 13–14; recursive commands and, 22–24, 39, 42; redirection and, 27–29, 33; regular expressions and, 165; relational databases and, 345, 350; relative path and, 21–23, 40; removing files or directories and, 19, 23–24, 27, 29, 32, 44, 46; renaming files or directories and, 23; reproducibility and, 13; scripting and, 43–47; sets and, 13, 30n7, 32; setting a path and, 49; shells and, 12–22, 29, 43–44, 49, 54; sorting and, 25–26, 31–35, 44–46; statistical computing and, 285; steep learning curve of, 13; strings and, 13, 15, 19, 27–28, 32, 38; Sun Solaris and, 12; syntax and, 49; terminals and, 12–18, 43, 45n12, 49; text editors and, 24, 32, 43, 50; text files and, 13–14, 24–27, 30, 32, 36, 43, 50; tips and tricks for, 49–51; Ubuntu and, 12–19, 43, 50; variables and, 15, 44, 47–51; version control and, 60–61, 66; wildcards and, 35–36, 39–40, 47; Windows and, 26, 43, 50 unpickled objects, 132–33

- Urban, M. C., 335–36
- user-defined functions, 83, 121–22, 275, 281, 310
- UTF-8, 113, 343
- utilities, 17, 209, 214
- variables, 370; adding, 309–10; coding and, 122–23, 127, 135, 137, 139–44, 151, 155–56, 158, 160–61; continuous, 321–24, 327, 385; data wrangling and, 304–10, 313–14, 319, 321–24, 327–29, 333; discrete, 321, 323, 327; Python and, 82–83, 87–91, 96, 102, 135, 137, 144, 155–60, 249, 278; R and, 249, 253–54, 257, 262–63, 270, 272–73, 277–78, 280, 284, 286–90; regular expressions and, 179–80, 214; Unix and, 15, 44, 47–51
- variable scope, 160–61, 278
- vectors: data structures and, 185–86, 195–96, 200, 253, 255–58, 261–62, 272–73, 276–83, 380–81; scientific computing and, 185–86, 195–96, 200, 253, 255–58, 261–62, 272–73, 276–83, 380–81; statistical computing and, 253, 255–58, 261–62, 272–73, 276–83
- verbose regular expression, 181
- version control: automation and, 55; branching and, 55, 70–78; bugs and, 66–67; centralized, 56–57; comma-separated values (CSV) format and, 61, 66; comments and, 72, 74–76; commits and, 55–79, 374; distributed, 57; documentation and, 74, 78n5; forks and, 78–79; Git and, 4, 56–73, 77–80; indexes and, 61, 64, 66; LaTeX and, 61; merging and, 70–79; modifications and, 56–57, 61, 63, 67, 75, 77–78; organization and, 4, 55–56; pull request and, 79; readability and, 61; reasons for using, 55–56; repositories and, 55–73, 77–80; reproducibility and, 56, 80; science and, 4, 78–79; shells and, 70n4, 71; simulations and, 61; staging and, 61, 63, 67–69, 73–74, 77; stashing and, 69–70; terminals and, 57–59, 62; text editors and, 57, 60–62, 65, 77; text files and, 61; Unix and, 60–61, 66; visualization and, 68
- violin plots, 320–23, 325n4
- visualization: clarity and, 318; coordinates and, 100, 165, 183, 291, 319, 326, 353, 355; data sets and, 300, 304–5, 309n2, 310, 314, 317, 323, 326; data wrangling and, 1, 5, 300, 318–33, 383–87; efficient data analysis and, 300; faceting and, 319, 328–29; filters and, 323–31; geometry and, 319–23, 333; ggplot2 and, 318–24, 327–28, 330; grammar of graphics and, 319–20; legends and, 330–31; mapping and, 319, 321, 326–27, 332, 386; plotting and, 300, 318–26, 333; R and, 82; regular expressions and, 183; saving and, 332–33; scales and, 318–19, 326–29; scientific computing and, 185, 191, 208; setting a feature and, 332; smoothing and, 319, 324, 333; themes and, 331–32; tidyverse and, 5, 300–4, 308, 309n2, 313, 318, 320, 336; version control and, 68; Wilkinson and, 319
- Wickham, Hadley, 300
- wildcards, 35–36, 39–40, 47, 349–50
- Wiles, Andrew, 7
- Wilkinson, Leland, 319
- Windows, 10; Command Prompt and, 15; Git Bash and, 13–15, 17, 21, 50, 136n3; line terminators and, 26; MiKTeX and, 222; Python and, 176n1; scientific typesetting and, 222; text editors and, 43; Unix and, 26, 43, 50
- word processors, 3, 220–21, 223
- workflow: Git and, 56, 58–64, 66, 69, 79; scientific computing and, 208; statistical computing and, 285
- XML. *See* Extensible Markup Language (XML)
- Żmihorski, M., 364
- z-score, 312–13