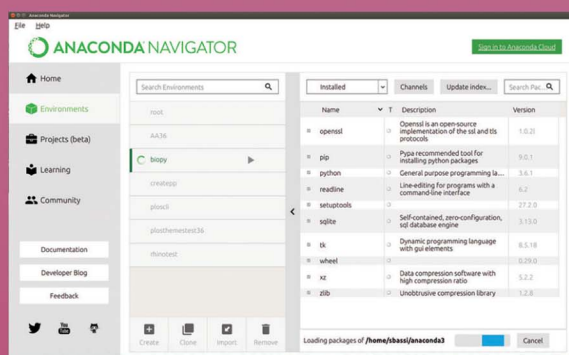
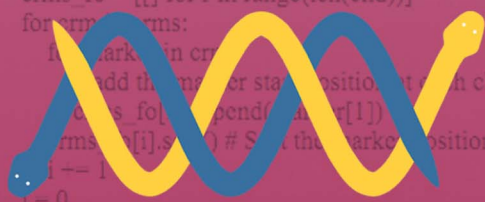


Chapman & Hall/CRC
Mathematical and Computational Biology Series

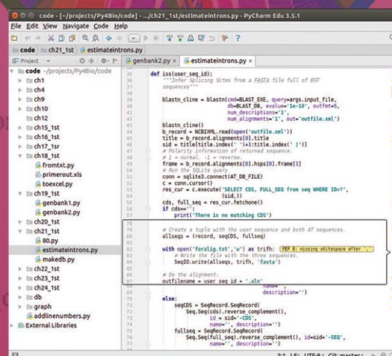
PYTHON FOR BIOINFORMATICS SECOND EDITION



```
def sortmarkers(crms, end):  
    """ Sort markers into chromosomes """  
    i = 0  
    crms_o = [[] for r in range(len(end))] # Sort the marker positions.  
    for crm in crms:  
        # Add the marker start position to the chromosome  
        # for which it depends on (if any).  
        crms_o[crms.index(crm)] # Sort the marker positions.  
        i += 1  
    return crms_o
```



biopython



SEBASTIAN BASSI

PYTHON FOR BIOINFORMATICS

SECOND EDITION

CHAPMAN & HALL/CRC

Mathematical and Computational Biology Series

Aims and scope:

This series aims to capture new developments and summarize what is known over the entire spectrum of mathematical and computational biology and medicine. It seeks to encourage the integration of mathematical, statistical, and computational methods into biology by publishing a broad range of textbooks, reference works, and handbooks. The titles included in the series are meant to appeal to students, researchers, and professionals in the mathematical, statistical and computational sciences, fundamental biology and bioengineering, as well as interdisciplinary researchers involved in the field. The inclusion of concrete examples and applications, and programming techniques and examples, is highly encouraged.

Series Editors

N. F. Britton

Department of Mathematical Sciences

University of Bath

Xihong Lin

Department of Biostatistics

Harvard University

Nicola Mulder

University of Cape Town

South Africa

Maria Victoria Schneider

European Bioinformatics Institute

Mona Singh

Department of Computer Science

Princeton University

Anna Tramontano

Department of Physics

University of Rome La Sapienza

Proposals for the series should be submitted to one of the series editors above or directly to:

CRC Press, Taylor & Francis Group

3 Park Square, Milton Park

Abingdon, Oxfordshire OX14 4RN

UK

Published Titles

**An Introduction to Systems Biology:
Design Principles of Biological Circuits**

Uri Alon

**Glycome Informatics: Methods and
Applications**

Kiyoko F. Aoki-Kinoshita

**Computational Systems Biology of
Cancer**

*Emmanuel Barillot, Laurence Calzone,
Philippe Hupé, Jean-Philippe Vert, and
Andrei Zinovyev*

Python for Bioinformatics, Second Edition

Sebastian Bassi

**Quantitative Biology: From Molecular to
Cellular Systems**

Sebastian Bassi

**Methods in Medical Informatics:
Fundamentals of Healthcare
Programming in Perl, Python, and Ruby**

Jules J. Berman

**Chromatin: Structure, Dynamics,
Regulation**

Ralf Blossey

**Computational Biology: A Statistical
Mechanics Perspective**

Ralf Blossey

Game-Theoretical Models in Biology

Mark Broom and Jan Rychtář

**Computational and Visualization
Techniques for Structural Bioinformatics
Using Chimera**

Forbes J. Burkowski

**Structural Bioinformatics: An Algorithmic
Approach**

Forbes J. Burkowski

Spatial Ecology

*Stephen Cantrell, Chris Cosner, and
Shigui Ruan*

**Cell Mechanics: From Single Scale-
Based Models to Multiscale Modeling**

*Arnaud Chauvière, Luigi Preziosi,
and Claude Verdier*

**Bayesian Phylogenetics: Methods,
Algorithms, and Applications**

Ming-Hui Chen, Lynn Kuo, and Paul O. Lewis

Statistical Methods for QTL Mapping

Zehua Chen

**An Introduction to Physical Oncology:
How Mechanistic Mathematical
Modeling Can Improve Cancer Therapy
Outcomes**

*Vittorio Cristini, Eugene J. Koay,
and Zhihui Wang*

**Normal Mode Analysis: Theory and
Applications to Biological and Chemical
Systems**

Qiang Cui and Ivet Bahar

Kinetic Modelling in Systems Biology

Oleg Demin and Igor Goryanin

Data Analysis Tools for DNA Microarrays

Sorin Draghici

**Statistics and Data Analysis for
Microarrays Using R and Bioconductor,
Second Edition**

Sorin Drăghici

**Computational Neuroscience:
A Comprehensive Approach**

Jianfeng Feng

**Biological Sequence Analysis Using
the SeqAn C++ Library**

Andreas Gogol-Döring and Knut Reinert

**Gene Expression Studies Using
Affymetrix Microarrays**

Hinrich Göhlmann and Willem Talloen

**Handbook of Hidden Markov Models
in Bioinformatics**

Martin Gollery

**Meta-analysis and Combining
Information in Genetics and Genomics**

Rudy Guerra and Darlene R. Goldstein

**Differential Equations and Mathematical
Biology, Second Edition**

D.S. Jones, M.J. Plank, and B.D. Sleeman

Knowledge Discovery in Proteomics

Igor Jurisica and Dennis Wigle

**Introduction to Proteins: Structure,
Function, and Motion**

Amit Kessel and Nir Ben-Tal

Published Titles (continued)

RNA-seq Data Analysis: A Practical Approach

*Eija Korpelainen, Jarno Tuimala,
Panu Somervuo, Mikael Huss, and Garry Wong*

Introduction to Mathematical Oncology

*Yang Kuang, John D. Nagy, and
Steffen E. Eikenberry*

Biological Computation

Ehud Lamm and Ron Unger

Optimal Control Applied to Biological Models

Suzanne Lenhart and John T. Workman

Clustering in Bioinformatics and Drug Discovery

John D. MacCuish and Norah E. MacCuish

Spatiotemporal Patterns in Ecology and Epidemiology: Theory, Models, and Simulation

*Horst Malchow, Sergei V. Petrovskii, and
Ezio Venturino*

Stochastic Dynamics for Systems Biology

Christian Mazza and Michel Benaïm

Statistical Modeling and Machine Learning for Molecular Biology

Alan M. Moses

Engineering Genetic Circuits

Chris J. Myers

Pattern Discovery in Bioinformatics: Theory & Algorithms

Laxmi Parida

Exactly Solvable Models of Biological Invasion

Sergei V. Petrovskii and Bai-Lian Li

Computational Hydrodynamics of Capsules and Biological Cells

C. Pozrikidis

Modeling and Simulation of Capsules and Biological Cells

C. Pozrikidis

Cancer Modelling and Simulation

Luigi Preziosi

Introduction to Bio-Ontologies

Peter N. Robinson and Sebastian Bauer

Dynamics of Biological Systems

Michael Small

Genome Annotation

*Jung Soh, Paul M.K. Gordon, and
Christoph W. Sensen*

Niche Modeling: Predictions from Statistical Distributions

David Stockwell

Algorithms for Next-Generation Sequencing

Wing-Kin Sung

Algorithms in Bioinformatics: A Practical Introduction

Wing-Kin Sung

Introduction to Bioinformatics

Anna Tramontano

The Ten Most Wanted Solutions in Protein Bioinformatics

Anna Tramontano

Combinatorial Pattern Matching Algorithms in Computational Biology Using Perl and R

Gabriel Valiente

Managing Your Biological Data with Python

*Allegra Via, Kristian Rother, and
Anna Tramontano*

Cancer Systems Biology

Edwin Wang

Stochastic Modelling for Systems Biology, Second Edition

Darren J. Wilkinson

Big Data Analysis for Bioinformatics and Biomedical Discoveries

Shui Qing Ye

Bioinformatics: A Practical Approach

Shui Qing Ye

Introduction to Computational Proteomics

Golan Yona

Chapman & Hall/CRC Mathematical and Computational Biology Series

PYTHON FOR BIOINFORMATICS

SECOND EDITION

SEBASTIAN BASSI



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2018 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed on acid-free paper
Version Date: 20170626

International Standard Book Number-13: 978-1-1380-3526-3 (Hardback)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Names: Bassi, Sebastian, author.
Title: Python for bioinformatics / Sebastian Bassi.
Description: Second edition. | Boca Raton : CRC Press, 2017. | Series: Chapman & Hall/CRC mathematical and computational biology | Includes bibliographical references and index.
Identifiers: LCCN 2017014460 | ISBN 9781138035263 (pbk. : alk. paper) | ISBN 9781138094376 (hardback : alk. paper) | ISBN 9781315268743 (ebook) | ISBN 9781351976961 (ebook) | ISBN 9781351976954 (ebook) | ISBN 9781351976947 (ebook)
Subjects: LCSH: Bioinformatics. | Python (Computer program language)
Classification: LCC QH324.2 .B387 2017 | DDC 570.285--dc23
LC record available at <https://lccn.loc.gov/2017014460>

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>
and the CRC Press Web site at
<http://www.crcpress.com>

Contents

List of Figures	xix
List of Tables	xxiii
Preface to the First Edition	xxv
Preface to the Second Edition	xxvii
Acknowledgments	xxxix
SECTION I Programming	
CHAPTER 1 ■ Introduction	3
1.1 WHO SHOULD READ THIS BOOK	3
1.1.1 What the Reader Should Already Know	4
1.2 USING THIS BOOK	4
1.2.1 Typographical Conventions	4
1.2.2 Python Versions	5
1.2.3 Code Style	5
1.2.4 Get the Most from This Book without Reading It All	6
1.2.5 Online Resources Related to This Book	7
1.3 WHY LEARN TO PROGRAM?	7
1.4 BASIC PROGRAMMING CONCEPTS	8
1.4.1 What Is a Program?	8
1.5 WHY PYTHON?	10
1.5.1 Main Features of Python	10
1.5.2 Comparing Python with Other Languages	11
1.5.3 How Is It Used?	14
1.5.4 Who Uses Python?	15
1.5.5 Flavors of Python	15
1.5.6 Special Python Distributions	16

1.6	ADDITIONAL RESOURCES	17
CHAPTER 2 ■ First Steps with Python		19
<hr/>		
2.1	INSTALLING PYTHON	20
2.1.1	Learn Python by Using It	20
2.1.2	Install Python Locally	20
2.1.3	Using Python Online	21
2.1.4	Testing Python	22
2.1.5	First Use	22
2.2	INTERACTIVE MODE	23
2.2.1	Baby Steps	23
2.2.2	Basic Input and Output	23
2.2.3	More on the Interactive Mode	24
2.2.4	Mathematical Operations	26
2.2.5	Exit from the Python Shell	27
2.3	BATCH MODE	27
2.3.1	Comments	29
2.3.2	Indentation	30
2.4	CHOOSING AN EDITOR	32
2.4.1	Sublime Text	32
2.4.2	Atom	33
2.4.3	PyCharm	34
2.4.4	Spyder IDE	35
2.4.5	Final Words about Editors	36
2.5	OTHER TOOLS	36
2.6	ADDITIONAL RESOURCES	37
2.7	SELF-EVALUATION	37
CHAPTER 3 ■ Basic Programming: Data Types		39
<hr/>		
3.1	STRINGS	40
3.1.1	Strings Are Sequences of Unicode Characters	41
3.1.2	String Manipulation	42
3.1.3	Methods Associated with Strings	42
3.2	LISTS	44
3.2.1	Accessing List Elements	45

3.2.2	List with Multiple Repeated Items	45
3.2.3	List Comprehension	46
3.2.4	Modifying Lists	47
3.2.5	Copying a List	49
3.3	TUPLES	49
3.3.1	Tuples Are Immutable Lists	49
3.4	COMMON PROPERTIES OF THE SEQUENCES	51
3.5	DICTIONARIES	54
3.5.1	Mapping: Calling Each Value by a Name	54
3.5.2	Operating with Dictionaries	56
3.6	SETS	58
3.6.1	Unordered Collection of Objects	59
3.6.2	Set Operations	60
3.6.3	Shared Operations with Other Data Types	62
3.6.4	Immutable Set: Frozenset	63
3.7	NAMING OBJECTS	63
3.8	ASSIGNING A VALUE TO A VARIABLE VERSUS BINDING A NAME TO AN OBJECT	64
3.9	ADDITIONAL RESOURCES	67
3.10	SELF-EVALUATION	68
CHAPTER 4 ■ Programming: Flow Control		69
4.1	IF-ELSE	69
4.1.1	Pass Statement	74
4.2	FOR LOOP	75
4.3	WHILE LOOP	77
4.4	BREAK: BREAKING THE LOOP	78
4.5	WRAPPING IT UP	80
4.5.1	Estimate the Net Charge of a Protein	80
4.5.2	Search for a Low-Degeneration Zone	81
4.6	ADDITIONAL RESOURCES	83
4.7	SELF-EVALUATION	83
CHAPTER 5 ■ Handling Files		85
5.1	READING FILES	86

5.1.1	Example of File Handling	87
5.2	WRITING FILES	89
5.2.1	File Reading and Writing Examples	90
5.3	CSV FILES	90
5.4	PICKLE: STORING AND RETRIEVING THE CONTENTS OF VARIABLES	94
5.5	JSON FILES	96
5.6	FILE HANDLING: OS, OS.PATH, SHUTIL, AND PATH.PY MODULE	98
5.6.1	path.py Module	100
5.6.2	Consolidate Multiple DNA Sequences into One FASTA File	102
5.7	ADDITIONAL RESOURCES	102
5.8	SELF-EVALUATION	103
CHAPTER 6 ■ Code Modularizing		105
6.1	INTRODUCTION TO CODE MODULARIZING	105
6.2	FUNCTIONS	106
6.2.1	Standard Way to Make Python Code Modular	106
6.2.2	Function Parameter Options	110
6.2.3	Generators	113
6.3	MODULES AND PACKAGES	114
6.3.1	Using Modules	115
6.3.2	Packages	116
6.3.3	Installing Third-Party Modules	117
6.3.4	Virtualenv: Isolated Python Environments	119
6.3.5	Conda: Anaconda Virtual Environment	121
6.3.6	Creating Modules	124
6.3.7	Testing Modules	125
6.4	ADDITIONAL RESOURCES	127
6.5	SELF-EVALUATION	128
CHAPTER 7 ■ Error Handling		129
7.1	INTRODUCTION TO ERROR HANDLING	129
7.1.1	Try and Except	131
7.1.2	Exception Types	134
7.1.3	Triggering Exceptions	135

7.2	CREATING CUSTOMIZED EXCEPTIONS	137
7.3	ADDITIONAL RESOURCES	138
7.4	SELF-EVALUATION	138
CHAPTER 8 ■ Introduction to Object Orienting Programming (OOP)		139
<hr/>		
8.1	OBJECT PARADIGM AND PYTHON	139
8.2	EXPLORING THE JARGON	140
8.3	CREATING CLASSES	142
8.4	INHERITANCE	145
8.5	SPECIAL METHODS	149
8.5.1	Create a New Data Type Using a Built-in Data Type	154
8.6	MAKING OUR CODE PRIVATE	154
8.7	ADDITIONAL RESOURCES	155
8.8	SELF-EVALUATION	156
CHAPTER 9 ■ Introduction to Biopython		157
<hr/>		
9.1	WHAT IS BIOPYTHON?	158
9.1.1	Project Organization	158
9.2	INSTALLING BIOPYTHON	159
9.3	BIOPYTHON COMPONENTS	162
9.3.1	Alphabet	162
9.3.2	Seq	163
9.3.3	MutableSeq	165
9.3.4	SeqRecord	166
9.3.5	Align	167
9.3.6	AlignIO	169
9.3.7	ClustalW	171
9.3.8	SeqIO	173
9.3.9	AlignIO	176
9.3.10	BLAST	177
9.3.11	Biological Related Data	187
9.3.12	Entrez	190
9.3.13	PDB	194
9.3.14	PROSITE	196
9.3.15	Restriction	197

9.3.16	SeqUtils	200
9.3.17	Sequencing	202
9.3.18	SwissProt	205
9.4	CONCLUSION	207
9.5	ADDITIONAL RESOURCES	207
9.6	SELF-EVALUATION	209

SECTION II Advanced Topics

CHAPTER 10 ■ Web Applications 213

10.1	INTRODUCTION TO PYTHON ON THE WEB	213
10.2	CGI IN PYTHON	214
10.2.1	Configuring a Web Server for CGI	215
10.2.2	Testing the Server with Our Script	215
10.2.3	Web Program to Calculate the Net Charge of a Protein (CGI version)	219
10.3	WSGI	221
10.3.1	Bottle: A Python Web Framework for WSGI	222
10.3.2	Installing Bottle	223
10.3.3	Minimal Bottle Application	223
10.3.4	Bottle Components	224
10.3.5	Web Program to Calculate the Net Charge of a Protein (Bottle Version)	229
10.3.6	Installing a WSGI Program in Apache	232
10.4	ALTERNATIVE OPTIONS FOR MAKING PYTHON-BASED DYNAMIC WEB SITES	232
10.5	SOME WORDS ABOUT SCRIPT SECURITY	232
10.6	WHERE TO HOST PYTHON PROGRAMS	234
10.7	ADDITIONAL RESOURCES	235
10.8	SELF-EVALUATION	236

CHAPTER 11 ■ XML 237

11.1	INTRODUCTION TO XML	237
11.2	STRUCTURE OF AN XML DOCUMENT	241
11.3	METHODS TO ACCESS DATA INSIDE AN XML DOCUMENT	246
11.3.1	SAX: cElementTree Iterparse	246

11.4	SUMMARY	251
11.5	ADDITIONAL RESOURCES	252
11.6	SELF-EVALUATION	252
CHAPTER 12 ■ Python and Databases		255
<hr/>		
12.1	INTRODUCTION TO DATABASES	256
12.1.1	Database Management: RDBMS	257
12.1.2	Components of a Relational Database	258
12.1.3	Database Data Types	260
12.2	CONNECTING TO A DATABASE	261
12.3	CREATING A MYSQL DATABASE	262
12.3.1	Creating Tables	263
12.3.2	Loading a Table	264
12.4	PLANNING AHEAD	266
12.4.1	PythonU: Sample Database	266
12.5	SELECT: QUERYING A DATABASE	269
12.5.1	Building a Query	271
12.5.2	Updating a Database	273
12.5.3	Deleting a Record from a Database	273
12.6	ACCESSING A DATABASE FROM PYTHON	274
12.6.1	PyMySQL Module	274
12.6.2	Establishing the Connection	274
12.6.3	Executing the Query from Python	275
12.7	SQLITE	276
12.8	NOSQL DATABASES: MONGODB	278
12.8.1	Using MongoDB with PyMongo	278
12.9	ADDITIONAL RESOURCES	282
12.10	SELF-EVALUATION	284
CHAPTER 13 ■ Regular Expressions		285
<hr/>		
13.1	INTRODUCTION TO REGULAR EXPRESSIONS (REGEX)	285
13.1.1	REGEX Syntax	286
13.2	THE RE MODULE	287
13.2.1	Compiling a Pattern	290
13.2.2	REGEX Examples	292

13.2.3	Pattern Replace	294
13.3	REGEX IN BIOINFORMATICS	294
13.3.1	Cleaning Up a Sequence	296
13.4	ADDITIONAL RESOURCES	297
13.5	SELF-EVALUATION	298
CHAPTER 14 ■ Graphics in Python		299
<hr/>		
14.1	INTRODUCTION TO BOKEH	299
14.2	INSTALLING BOKEH	299
14.3	USING BOKEH	301
14.3.1	A Simple X-Y Plot	303
14.3.2	Two Data Series Plot	304
14.3.3	A Scatter Plot	306
14.3.4	A Heatmap	309
14.3.5	A Chord Diagram	309
14.4	ADDITIONAL RESOURCES	311
SECTION III Python Recipes with Commented Source Code		
CHAPTER 15 ■ Sequence Manipulation in Batch		315
<hr/>		
15.1	PROBLEM DESCRIPTION	315
15.2	PROBLEM ONE: CREATE A FASTA FILE WITH RANDOM SE- QUENCES	315
15.2.1	Commented Source Code	315
15.3	PROBLEM TWO: FILTER NOT EMPTY SEQUENCES FROM A FASTA FILE	316
15.3.1	Commented Source Code	317
15.4	PROBLEM THREE: MODIFY EVERY RECORD OF A FASTA FILE	319
15.4.1	Commented Source Code	320
CHAPTER 16 ■ Web Application for Filtering Vector Contamination		321
<hr/>		
16.1	PROBLEM DESCRIPTION	321
16.1.1	Commented Source Code	322
16.2	ADDITIONAL RESOURCES	326
CHAPTER 17 ■ Searching for PCR Primers Using Primer3		329
<hr/>		

17.1	PROBLEM DESCRIPTION	329
17.2	PRIMER DESIGN FLANKING A VARIABLE LENGTH REGION	330
17.2.1	Commented Source Code	331
17.3	PRIMER DESIGN FLANKING A VARIABLE LENGTH REGION, WITH BIOPYTHON	332
17.4	ADDITIONAL RESOURCES	333
CHAPTER 18 ■ Calculating Melting Temperature from a Set of Primers		335
<hr/>		
18.1	PROBLEM DESCRIPTION	335
18.1.1	Commented Source Code	336
18.2	ADDITIONAL RESOURCES	336
CHAPTER 19 ■ Filtering Out Specific Fields from a GenBank File		339
<hr/>		
19.1	EXTRACTING SELECTED PROTEIN SEQUENCES	339
19.1.1	Commented Source Code	339
19.2	EXTRACTING UPSTREAM REGIONS OF SELECTED GENES	340
19.2.1	Commented Source Code	340
19.3	ADDITIONAL RESOURCES	341
CHAPTER 20 ■ Inferring Splicing Sites		343
<hr/>		
20.1	PROBLEM DESCRIPTION	343
20.1.1	Infer Splicing Sites with Commented Source Code	345
20.1.2	Sample Run of Estimate Intron Program	347
CHAPTER 21 ■ Web Server for Multiple Alignment		349
<hr/>		
21.1	PROBLEM DESCRIPTION	349
21.1.1	Web Interface: Front-End. HTML Code	349
21.1.2	Web Interface: Server-Side Script. Commented Source Code	351
21.2	ADDITIONAL RESOURCES	353
CHAPTER 22 ■ Drawing Marker Positions Using Data Stored in a Database		355
<hr/>		
22.1	PROBLEM DESCRIPTION	355
22.1.1	Preliminary Work on the Data	355

22.1.2	MongoDB Version with Commented Source Code	358
CHAPTER 23 ■ DNA Mutations with Restrictions		363
<hr/>		
23.1	PROBLEM DESCRIPTION	363
23.1.1	Introduce Point Mutations and Get Restriction Profile	363
23.1.2	Running Introduce Point Mutations Program	367
23.2	ADDITIONAL RESOURCES	368
 SECTION IV Appendices		
APPENDIX A ■ Collaborative Development: Version Control with GitHub		371
<hr/>		
A.1	INTRODUCTION TO VERSION CONTROL	372
A.2	VERSION YOUR CODE	373
A.3	SHARE YOUR CODE	380
A.4	CONTRIBUTE TO OTHER PROJECTS	386
A.5	CONCLUSION	387
A.6	METHODS	387
A.7	ADDITIONAL RESOURCES	387
 APPENDIX B ■ Install a Bottle App in PythonAnywhere		393
<hr/>		
B.1	PYTHONANYWHERE	393
B.1.1	What Is PythonAnywhere	393
B.1.2	Installing a Web App in PythonAnywhere	393
 APPENDIX C ■ Scientific Python Cheat Sheet		403
<hr/>		
C.1	PURE PYTHON	404
C.2	VIRTUALENV	410
C.3	CONDA	412
C.4	IPYTHON	413
C.5	NUMPY	415
C.6	MATPLOTLIB	420
C.7	SCIPY	422
C.8	PANDAS	423

Index	427
-------	-----



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

List of Figures

2.1	Anaconda install in macOS.	21
2.2	Anaconda Python interactive terminal.	23
2.3	PyCharm Edu welcome screen.	35
3.1	Intersection.	60
3.2	Union.	61
3.3	Difference.	61
3.4	Symmetric difference.	62
3.5	Case 1.	65
3.6	Case 2.	66
5.1	Excel formatted spreadsheet called <code>sampladata.xlsx</code> .	93
8.1	IUPAC nucleic acid notation table.	147
9.1	Anatomy of a BLAST result.	181
10.1	Our first CGI.	216
10.2	CGI accessed from local disk instead from a web server.	217
10.3	<code>greeting.html</code> : A very simple form.	217
10.4	Output of CGI program that processes <code>greeting.html</code> .	218
10.5	Form <code>protcharge.html</code> ready to be submitted.	220
10.6	Net charge CGI result.	222
10.7	Hello World program made in Bottle, as seen in a browser.	224
10.8	Form for the web app to calculate the net charge of a protein.	229
11.1	Screenshot of XML viewer.	244
11.2	Codebeautify, a web based XML viewer.	245
12.1	Screenshot of PhpMyAdmin.	258
12.2	Creating a new database using phpMyAdmin.	262
12.3	Creating a new table using phpMyAdmin.	264

12.4	View of the Student table.	266
12.5	An intentionally faulty “Grades” table.	267
12.6	A better “Grades” table.	267
12.7	Courses table: A lookup table.	268
12.8	Modified “Grades” table.	268
12.9	Screenshot of SQLite manager.	277
12.10	View from a MongoDB cloud provider.	281
14.1	A circle with Bokeh.	302
14.2	Four circles with Bokeh.	303
14.3	A simple plot with Bokeh.	305
14.4	A two data series plot with Bokeh.	306
14.5	Scatter plot graphics.	308
14.6	A heatmap out of a microarray experiment.	310
14.7	A chord diagram.	312
16.1	HTML form for sequence filtering.	327
16.2	HTML form for sequence filtering.	328
21.1	Muscle Web interface.	350
22.1	Product of Listing 22.2, using the demo dataset (NODBDEMO).	356
A.1	The git add/commit process.	375
A.2	Working with a local repository.	389
A.3	Working with both a local and remote repository as a single user.	390
A.4	Contributing to open source projects.	391
B.1	“Consoles” tab.	394
B.2	The “Web” tab.	395
B.3	Upgrading domain type option.	396
B.4	Select a web framework screen, select Bottle.	397
B.5	Select a Python and Bottle version.	398
B.6	Form to enter the path of the web app.	399
B.7	The sample web app is ready to use.	399
B.8	The “File” tab.	400
B.9	Form to create a new directory in PythonAnywhere.	400
B.10	View and upload files into your account.	400

B.11	Front-end of the program to calculate charge of a protein using Bottle and hosted in PythonAnywhere.	401
------	--	-----



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

List of Tables

2.1	Arithmetic-Style Operators	26
3.1	Common List Operations	48
3.2	Methods Associated with Dictionaries	58
9.1	Sequence and Alignment Formats	175
9.2	Blast programs	178
9.3	eUtils	191
10.1	Frameworks for Web Development	233
12.1	Students in Python University	259
12.2	Table with primary key	260
12.3	MySQL Data Types	261
13.1	REGEX Special Sequences	287
A.1	Resources	373



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Preface to the First Edition

This book is a result of the experience accumulated during several years of working for an agricultural biotechnology company. As a genomic database curator, I gave support to staff scientists with a broad range of bioinformatics needs. Some of them just wanted to automate the same procedure they were already doing by hand, while others would come to me with biological problems to ask if there were bioinformatics solutions. Most cases had one thing in common: Programming knowledge was necessary for finding a solution to the problem. The main purpose of this book is to help those scientists who want to solve their biological problems by helping them to understand the basics of programming. To this end, I have attempted to avoid taking for granted any programming-related concepts. The chosen language for this task is Python.

Python is an easy-to-learn computer language that is gaining traction among scientists. This is likely because it is easy to use, yet powerful enough to accomplish most programming goals. With Python the reader can start doing real programming very quickly. Journals such as *Computing in Science and Engineering*, *Briefings in Bioinformatics*, and *PLOS Computational Biology* have published introductory articles about Python. Scientists are using Python for molecular visualization, genomic annotation, data manipulation, and countless other applications.

In the particular case of the life sciences, the development of Python has been very important; the best exponent is the Biopython package. For this reason, Section II is devoted to Biopython. Anyhow, I don't claim that Biopython is the solution to every biology problem in the world. Sometimes a simple custom-made solution may better fit the problem at hand. There are other packages like BioNEB and CoreBio that the reader may want to try.

The book begins from the very basic, with Section I ("Programming"), teaching the reader the principles of programming. From the very beginning, I place a special emphasis on practice, since I believe that programming is something that is best learned by doing. That is why there are code fragments spread over the book. The reader is expected to experiment with them, and attempt to internalize them. There are also some spare comparisons with other languages; they are included only when doing so enlightens the current topic. I believe that most language comparisons do more harm than good when teaching a new language. They introduce information that is incomprehensible and irrelevant for most readers.

In an attempt to keep the interest of the reader, most examples are somehow related to biology. In spite of that, these examples can be followed even if the reader doesn't have any specific knowledge in that field.

To reinforce the practical nature of this book, and also to use as reference

material, Section IV is called “Python Recipes with Commented Source Code.” These programs can be used as is, but are intended to be used as a basis for other projects. Readers may find that some examples are very simple; they do their job without too many bells and whistles. This is intentional. The main reason for this is to illustrate a particular aspect of the application without distracting the reader with unnecessary features, as well as to avoid discouraging the reader with complex programs. There will always be time to add features and customizations once the basics have been learned.

The title of Section III (“Advanced Topics”) may seem intimidating, but in this case, advanced doesn’t necessarily mean difficult. Eventually, everyone will use the chapters in this section [especially relational database management system—RDBMS— and XML]. An important part of the bioinformatics work is building and querying databases, which is why I consider knowing a RDBMS like MySQL to be a relevant part of the bioinformatics skill set. Integrating data from different sources is one of tasks most frequently performed in bioinformatics. The tool of choice for this task is XML. This standard is becoming a widely used platform for data interchange between applications. Python has several XML parsers and we explain most of them in this book.

Appendix B, “Selected Papers,” provides introductory level papers on Python. Although there is some overlapping of subjects, this was done to show several points of view of the same subject.

Researchers are not the only ones for whom this book will be beneficial. It has also been structured to be used as a university textbook. Students can use it for programming classes, especially in the new bioinformatics majors.

Preface to the Second Edition

The first edition of *Python for Bioinformatics* was written in 2008 and published in 2009. Even after eight years, the lessons in this book are still valuable. This is quite an accomplishment in a field that evolves at such a fast pace. In spite of its usefulness, the book is showing its age and would greatly benefit from a second edition.

The predominant Python version is 3.6, although Python 2.7 is still in use in production systems. Since there are incompatibilities between these versions, lot of effort was made to make all code in the book Python 3 compatible.

Not only has the software changed in these past eight years, but enterprise attitude and support toward Open Source Software in general and Python in particular has changed dramatically. There are also new computing paradigms that can't be ignored such as collaborative development and cloud computing.

In the original book, Chapter 14 was called “Collaborative Development: Version Control” and was based on **Bazaar**, a software that follows the currently used distributed development workflow but is not what is being used by most developers today. By far the most software development is done with **Git** at **GitHub**. This chapter was rewritten to focus on current practices.

Web development is another area that changed significantly. Although this is not a book about web development, the chapter “Web Applications” now reflects current usage of long-running processes and frameworks instead of CGI/WSGI and middleware-based applications. Frameworks were discussed as a side note in this chapter, but now the chapter is based around a framework (**Bottle**) and leave the old method as a historical footnote.

In databases, the **NoSQL** gained lot of traction, from being a bullet point in the first edition, now has its own section using **MongoDB**, and a Python recipe was changed to use this NoSQL database.

Graphical libraries have improved since 2009, and there are great quality competing graphic libraries available for Python. There is a whole chapter devoted to **Bokeh**, a free interactive visualization library.

Another change that is reflected in this book is the usage of **Anaconda** and **Jupyter Notebooks** (with all code in a cloud notebook provided by Microsoft Azure¹).

¹See <https://notebooks.azure.com/py4bio/libraries/py3.us>

Regarding source code, there is a GitHub repository at <https://github.com/Serulab/Py4Bio> where you can download all the code and sample files used in this book.

There are corrections in every chapter. Sometimes there were actual mistakes, but most of the corrections were related to the Python 3 upgrade and in keeping with current good practices. Regarding corrections, I expect that this book may need corrections, so I made a web page where the readers can get updates. Please take a look at <http://py3.us> and subscribe to the low volume mailing list while at it.

Apart from software evolution and paradigms shifts, I also gained development experience and changed my views on pedagogical matters. During these years I worked in a genome sequencing project at an international consortium and as a senior software developer in an NYSE listed company (Globant). In the last 5 years I worked for several well-known clients such as Salesforce and National Geographic. I am currently working at PLOS (Public Library of Science).

By request of MATLAB, I include their contact information:

MATLAB ® is a registered trademark of The MathWorks, Inc. For product information please contact: The MathWorks, Inc. 3 Apple Hill Drive Natick, MA, 01760-2098 USA Tel: 508-647-7000 Fax: 508-647-7001 E-mail: info@mathworks.com Web: www.mathworks.com

Regarding the logo of Biopython, that is used in the cover, here it is usage license (this covers all Biopython files, including its logo):

Biopython is currently released under the "Biopython License Agreement" (given in full below). Unless stated otherwise in individual file headers, all Biopython's files are under the "Biopython License Agreement".

Some files are explicitly dual licensed under your choice of the "Biopython License Agreement" or the "BSD 3-Clause License" (both given in full below). This is with the intention of later offering all of Biopython under this dual licensing approach.

Biopython License Agreement

Permission to use, copy, modify, and distribute this software and its documentation with or without modifications and for any purpose and without fee is hereby granted, provided that any copyright notices appear in all copies and that both those copyright notices and this permission notice appear in supporting documentation, and that the names of the contributors or copyright holders not be used in advertising or publicity pertaining to distribution of the software without specific prior permission.

THE CONTRIBUTORS AND COPYRIGHT HOLDERS OF THIS SOFTWARE DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING

FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

BSD 3-Clause License

Copyright (c) 1999-2017, The Biopython Contributors All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Acknowledgments

A project such as this book couldn't be done by just one person. For this reason, there is a long list of people who deserve my thanks. In spite of the fact that the average reader doesn't care about the names, and at the risk of leaving someone out, I would like to acknowledge the following people: my wife Virginia Gonzalez (Vicky) and my son Maximo Bassi, who had to contend with my virtual absence during more than a year. Vicky also assisted me in uncountable ways during manuscript preparation. My parents and professors taught me important lessons. My family (Oscar, Graciela, and Ramiro) helped me with the English copyediting, along with Hugo and Lucas Bejar. Vicky, Griselda, and Eugenio also helped by providing a development abstraction layer, which is needed for writers and developers.

I would like to thank the people in the local Python community (<http://www.python.org.ar>): Facundo Batista, Lucio Torre, Gabriel Genellina, John Lenton, Alejandro J. Cura, Manuel Kaufmann, Gabriel Patiño, Alejandro Weil, Marcelo Fernandez, Ariel Rossanigo, Mariano Draghi, and Buanzo. I would choose Python again just for this great community. I also thank the people at Biopython: Jeffrey Chang, Brad Chapman, Peter Cock, Michiel de Hoon, and Iddo Friedberg. Peter Cock is specially thanked for his comments on the Biopython chapter. I also thank Shashi Kumar and Pablo Di Napoli who helped me with the L^AT_EX₂_ε issues, and Sunil Nair who believed in me from the first moment. Also people at Globant who trusted in me, like Guido Barosio, Josefina Chausovsky, Lucas Campos, Pablo Brenner and Guibert Englebienne. Globant co-workers such as Pedro Mourelle, Chris DeBlois, Rodrigo Obi-Wan Iloro, Carlos Del Rio and Alejandro Valle. People at PLOS, Jeffrey Gray and Nick Peterson.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

I

Programming



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Introduction

CONTENTS

1.1	Who Should Read This Book	3
1.1.1	What the Reader Should Already Know	4
1.2	Using this Book	4
1.2.1	Typographical Conventions	4
1.2.2	Python Versions	5
1.2.3	Code Style	5
1.2.4	Get the Most from This Book without Reading It All	6
1.2.5	Online Resources Related to This Book	7
1.3	Why Learn to Program?	7
1.4	Basic Programming Concepts	8
1.4.1	What Is a Program?	8
1.5	Why Python?	10
1.5.1	Main Features of Python	10
1.5.2	Comparing Python with Other Languages	11
	Readability	12
	Speed	13
1.5.3	How Is It Used?	14
1.5.4	Who Uses Python?	15
1.5.5	Flavors of Python	15
1.5.6	Special Python Distributions	16
1.6	Additional Resources	17

The most effective way to do it, is to do it.

Amelia Earhart

1.1 WHO SHOULD READ THIS BOOK

This book is for the life science researcher who wants to learn how to program. He/she may have previous exposure to computer programming, but this is not necessary to understand this book (although it surely helps).

This book is designed to be useful to several separate but related audiences, students, graduates, postdocs, and staff scientists, since all of them can benefit from knowing how to program.

Exposing students to programming at early stages in their career helps to boost their creativity and logical thinking, and both skills can be applied in research. In order to ease the learning process for students, all subjects are introduced with the minimal prerequisites. There are also questions at the end of each chapter. They can be used for self-assessing how much you've learned. The answers are available to teachers in a separate guide.

Graduates and staff scientists having actual programming needs should find its several real-world examples and abundant reference material extremely valuable.

1.1.1 What the Reader Should Already Know

Since this book is called *Python for Bioinformatics*, it has been written with the following assumptions in mind:

- No programming knowledge is assumed, but the reader is required to have minimum computer proficiency to be able to use a text editor and handle basic tasks in your operating system (OS). Since Python is multi-platform, most instructions from this book will apply to the most common operating systems (Windows, macOS and Linux); when there is a command or a procedure that applies only to a specific OS, it will be clearly noted.
- The reader should be working (or at least planning to work) with bioinformatics tools. Even low-scale handmade jobs, such as using the NCBI BLAST to ID a sequence, aligning proteins, primer searching, or estimating a phylogenetic tree will be useful to follow the examples. The more familiar the reader is with bioinformatics, the better he will be able to apply the concepts learned in this book.

1.2 USING THIS BOOK

1.2.1 Typographical Conventions

There are some typographical conventions I have tried to use in a uniform way throughout the book. They should aid readability and were chosen to tell apart user-made names (or variables) from language keywords. This comes in handy when learning a new computer language.

Bold: Objects provided by Python and by third-party modules. With this notation it should be clear that **round** is part of the language and not a user-defined name. Bold is also used to highlight parts of the text. **There is no way** to confuse one bold usage with the other.

Mono-spaced font: User declared variables, code, and filenames. For example: `sequence = 'MRVLLVALALLALAASATS'`.

Italics: In commands, it is used to denote a variable that can take different values. For example, in `len(iterable)`, “iterable” can take different values. Used in

text, it marks a new word or concept. For example “One such fundamental data structure is a *dictionary*.”

The content of lines starting with \$ (dollar sign) are meant to be typed in your operating system console (also called **command prompt** in Windows or **terminal** in macOS).

↵ : Break line. Some lines are longer than the available space in a printed page, so this symbol is inserted to mean that what is on the next line in the page represents the same line on the computer screen. Inside code, the symbol used is `<=`.

1.2.2 Python Versions

The current version of Python at this moment is 3.6.1. There is a 2.7.12 version that is maintained¹ because there are still a sizable number of applications in production using the 2.7 branch. Versions 3.x and 2.x are slightly different, at the point of being incompatible. Python 3 is more efficient than Python 2 in many aspects. Large websites such as Instagram migrated from Python 2.7 to Python 3.6 to save in CPU and memory consumption by up to 30%. This book uses Python 3.6.

The only scenario where you may need to use Python 2.7, apart from maintenance of old code, is when there is no availability of a specific library for Python 3. In this case, before starting a project in Python 2.7, try to search for a replacement library. For example, you want to connect with a MySQL database and you are told to use **MySQLdb**, since this package is not Python 3 compatible; instead of using Python 2.7, use **mysqlclient** or **mysql-connector-python**, both works with Python 3.

1.2.3 Code Style

Python source code that appears in this book is presented as **listings**. Each line of these listings is numbered. These numbers are not intended to be typed; they are used to reference each line in the text. You don't need to copy the code from the book, since it can be downloaded from the GitHub repository at <https://github.com/Serulab/Py4Bio>.

Code can be formatted in several ways and still be valid to the Python interpreter. This following code is syntactically correct:

```
def GetAverage(X):
    avG=sum(X)/len(X)
    " Calculate the average "
    return avG
```

Also this one:

¹Python 2.7.x has an end-of-life date in 2020. There will be no Python 2.8. For more information see <https://www.python.org/dev/peps/pep-0373/>.

```
def get_average(items):
    """ Calculate the average
    """
    average = sum(items) / len(items)
    return average
```

The former code sample follows most accepted coding styles for Python.² Throughout the book you will find mostly code formatted as the second sample. Some code in the book will not follow accepted coding styles for the following reasons:

- There are some instances where the most didactic way to show a particular piece of code conflicts with the style guide. On those few occasions, I choose to deviate from the style guide in favor of clarity.
- Due to size limitation in a printed book, some names were shortened and other minor drifts from the coding styles have been introduced.
- To show that there is more than one way to write the same code. Coding style is a guideline, and enforcement is not made at a language level, so some programmers don't follow it thoroughly. You should be able to read "bad" code, since sooner or later you will have to read other people's code.

1.2.4 Get the Most from This Book without Reading It All

- If you want to **learn how to program**, read the first section, from Chapter 1 to Chapter 8. The Regular Expressions (REGEX) chapter (Chapter 13) can be skipped if you don't need to deal with REGEX.
- If you know Python and just want to **know about Biopython**, read first Chapter 9 (from page 158 to page 209). It is about Biopython modules and functions. Then try to follow programs found in Section III (from page 315 to page 369).
- There are three appendixes that can be read in an independent way. Appendix A (Collaborative Development: Version Control with GitHub) reproduces a paper called "A Quick Introduction to Version Control with Git and GitHub." Appendix B shows how to install a web application using Python Anywhere. Appendix C is a reference material that can be used as a cheat sheet when you need a quick answer without having to read a chapter.

²The official Python style guide is located at <https://www.python.org/dev/peps/pep-0008>, and a more easy-to-read style guide is located at <http://docs.python-guide.org/en/latest/writing/style>.

1.2.5 Online Resources Related to This Book

The book website is at <http://py3.us>. In this site you will find errata, a mailing list to keep updated about Python and links to source code repositories. Regarding source code, the official source code repository of this book is at GitHub (<https://github.com/Serulab/Py4Bio>). From this site you can inspect online or download all the code used in this book. To download all scripts, go to the “Clone or download” green button and press it. If you have Git installed in your machine (and know how to use it³), clone the repository using this address: `git@github.com:Serulab/Py4Bio.git`. Another alternative is to click on “Download ZIP”. Once you have the repository in your machine, go to the code folder, where there are a set of folders, each one has the scripts related to the chapter. Each script in the book has a name and this corresponds with the filename. There is another folder called `notebooks`, and it contains Jupyter notebooks that can be run locally. For more information on how to run a Jupyter notebook, please see <http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/execute.html>.

Another online resource are the Jupyter Notebooks available at Microsoft Azure Notebook website (<https://notebooks.azure.com/py4bio/libraries/py3.us>). The same notebooks that are in the book repository, can be used online in this site.

1.3 WHY LEARN TO PROGRAM?

Many of the tasks that a researcher performs with his or her computer are repetitive: Collect data from a Web page, convert files from one format to another, execute or interpret hundreds of BLAST results, primer design, look for restriction enzymes, etc. In many cases it is evident that these are tasks that can be performed with a computer, with less effort on our part and without the possibility of errors caused by tiredness or distractions.

An important consideration when you’re evaluating whether or not to create a program is the apparent time lost in the definition and formulation of the problem, implementing it with code, and then debugging it (correcting errors). It is incorrect to consider problem definition and evaluation as a waste of time. It is generally at this precise point in the process where we understand thoroughly the problem that we face. It is common that during the attempt to formulate a problem, we realize that many of our initial assumptions were mistaken. It also helps us to detect when it is necessary to restart the planning process. When this happens, it is better that it happens at the planning stage than when we are in the middle of the project. In these cases, the planning of the program represents time saved. Another advantage to take into account is that the time that is invested to create a program once is compensated by the speed with which the tasks are performed every time we run it.

³In Appendix A there is a tutorial on how to use GitHub

Not only can it automate the procedures that we do manually, but it will also be able to do things that would otherwise not be possible.

Sometimes it is not very clear if a particular task can be done by a program. Reading a book such as this one (including the examples) will help you identify which tasks are feasible to automate with software and which ones are better done manually.

1.4 BASIC PROGRAMMING CONCEPTS

Before installing Python, let's review some programming fundamentals. If you have some previous programming experience, you may want to skip this section and jump straight to Chapter 2 "Installing Python." This section introduces basic concepts such as *instructions*, *data types*, *variables*, and some other related terminology that is used throughout this book.

1.4.1 What Is a Program?

Computers only know what you tell them. The way to tell them to do something is by a program. A *program* is a set of ordered instructions designed to command the computer to do something. The word "ordered" is there because is not enough to declare what to do, but the actual order of directions should also be stated.⁴

A program is often characterized as a recipe. A typical recipe consists of a list of ingredients followed by step-by-step instructions on how to prepare a dish. This analogy is reflected in several programming websites and tutorials with the words "recipe" and "cookbook." A laboratory protocol is another useful analogy. A protocol is defined as a "predefined written procedural method in the design and implementation of experiments."

Here is a typical protocol, followed almost every day in several molecular laboratories:

Listing 1.1: Protocol for Lambda DNA digestion

Restriction Digestion of Lambda DNA

Materials

5.0 mcL	Lambda DNA (0.1 g/L)
2.5 mcL	10x buffer
16.5 mcL	H ₂ O
1.0 mcL	EcoRI

⁴There are *declarative* languages that state what the program should accomplish, rather than describing how to accomplish it. Most computer languages (Python included) are *imperative* instead of *declarative*.

Procedure

Incubate the reagents at 37°C for 1 hr.

Add 2.5 mcL loading dye and incubate for another 15 minutes.

Load 20 mcL of the digestion mixture onto a minigel

There are at least two components of a protocol: materials or ingredients, and procedures. A procedure provides specific order like **incubate**, add, mix, **load** and many others. The same goes for a computer program. The programmer gives specific order to the computer: **print**, **read**, **write**, **add**, **multiply**, **round**, and others.

While protocol procedures correlate with program instructions, materials are the *data*. In protocols, procedures are applied to materials: Mix 2.5 μL of buffer with 5 μL of Lambda DNA and 16.5 μL of H_2O , load 20 μL onto a minigel. In a program, instructions are applied to data: print the text string “Hello”, add two integer numbers, round a float number.

As a protocol can be written in different languages (like English, Spanish, or French), there are different languages to program a computer. In science, English is the *de facto* language. Due to historical, commercial and practical reasons, there is no such equivalent in computer science. There are several languages, each with its own strong points and weakness. For reasons that will make sense shortly, Python was the computer language chosen for this book.

Let’s see a simple Python program:

Listing 1.2: sample.py: Sample Python Program

```
1 seq_1 = 'Hello,'
2 seq_2 = ' you!'
3 total = seq_1 + seq_2
4 seq_size = len(total)
5 print(seq_size)
```

Note: The numbers at the beginning of the each line are for reference only, they are not meant to be typed.

This small program can be read as “Name the string **Hello**, as **seq_1**. Name the string **you!** as **seq_2**. Add the strings named **seq_1** and **seq_2** and call the result as **total**. Get the length of the string called **total** and name this value as **seq_size**. Print the value of **seq_size**.” This program prints the number 11.

As shown, there are different types of data (often called “data types” or just “types”). Numbers (integers or float), text string, and other data types are covered in Chapter 3. In `print(seq_size)`, the instruction is **print** and **seq_size** is the name of the data. Data is often represented as *variables*. A *variable* is a name that stands for a value that may vary during program execution. With variables, a programmer can represent a generic command like “round n” instead of “round 2.9.” This way he can take into account a non-fixed (hence *variable*) value. When

the program is executed, “n” should take a specific value since there is no way to “round n.” This can be done by assigning a value to a variable or by binding a name to a value.⁵ The difference between “assign a value to a variable” and “bind a name to a value” is explained in detail in Chapter 3 (from page 64). In any case, it is expressed as:

```
variable_name = value
```

Note that **this is not an equality** as seen in mathematics. In an equality, terms can be interchanged, but in programming, the term on the right (**value**) takes the name of the term on the left (**variable_name**). For example,

```
seq_1 = 'Hello'
```

After this assignment, the variable `seq_1` can be used, like,

```
print(seq_1)
```

This is translated as “print out the value called `seq_1`”. This command returns “Hello” because this is the value of this variable.

1.5 WHY PYTHON?

Let’s have a look at some Python features worth pointing out.

1.5.1 Main Features of Python

- **Readability:** When we talk about readability, we refer as much to the original programmer as any other person interested in understanding the code. It is not an uncommon occurrence for someone to write some code then return to it a month later and find it difficult to understand. Sometimes Python is called a “human-readable language.”
- **Built-in features:** Python comes with “batteries included.” It has a rich and versatile standard library that is immediately available, without the user having to download separate packages. With Python you can, with few lines, read and write XML and JSON files, parse and generate email messages, extract files from a zip archive, open a URL as if were a file, and many other possibilities that in other languages, it would require a third-party library.
- **Availability of third-party modules for a broad spectrum of activities.** Data visualization⁶ and plotting, PDF generation, bioinformatics analysis,⁷ image

⁵In Python the latter form is used.

⁶Matplotlib (<http://matplotlib.org/>) and Bokeh <http://bokeh.pydata.org/en/latest/> are the most used.

⁷Biopython library to make your own bioinformatics applications (<http://biopython.org/>).

processing,⁸ machine learning,⁹ game development, interface with popular databases,¹⁰ and application software are only a handful of examples of modules that can be installed to extend Python functionality.

- High-level built-in data structures: Dictionaries, sets, lists, tuples, and others. These are very useful to model real-world data. Third-party modules such as NumPy and SciPy can also extend the structures to kd-trees, n-dimensional arrays, matrix operations, time-series, image objects, and more.
- Multiparadigm: Python can be used as a “classic” procedural language or as “modern” object-oriented programming (OOP) language. Most programmers start writing code in a procedural way and when they need to, they upgrade to OOP. Python doesn’t force programmers to write OOP code when they just want to write a simple script.
- Extensibility: If the built-in methods and available third-party modules are not enough for your needs, you can easily extend Python, even in other programming languages. There are some applications written mostly in Python but with a processor demanding routine in C or FORTRAN. Python can also be extended by connecting it to specialized high-level languages like R or MATLAB¹¹.
- Open source: Python has a liberal open source license that makes it freely usable and distributable, even for commercial use.
- Cross platform: A program made in Python can be run under any computer that has a Python interpreter. This way, a program made under Windows 10 can run unmodified in Linux or OSX. Python interpreters are available for most computer and operating systems, and even some devices with embedded computers like the Raspberry Pi.
- Thriving community: Python is nowadays the programming language to use for scientists and researchers.¹² This translates into more libraries for your projects and people you can go to for support.

1.5.2 Comparing Python with Other Languages

You may be wondering why you should use Python, and not more well-known languages like Java, PHP, or C++. It is a good question. A programming language

⁸Scikit-image paper: <http://peerj.com/articles/453>

⁹scikit-learn website: <http://scikit-learn.org/stable/>

¹⁰<https://wiki.python.org/moin/DatabaseProgramming>

¹¹MATLAB® is a registered trademark of The MathWorks, Inc. For product information please contact: The MathWorks, Inc. 3 Apple Hill Drive Natick, MA, 01760-2098 USA. Tel: 508-647-7000. Fax: 508-647-7001. E-mail: info@mathworks.com. Web: www.mathworks.com.

¹²<http://www.nature.com/news/programming-pick-up-python-1.16833>

can be regarded as a tool, and choosing the best tool for the job makes a lot of sense.

Readability

Nonprofessional programmers tend to value the learning curve as much as the legibility of the code (both aspects are tightly related).

A simple “hello world” program in Python looks like this:

```
print("Hello world!")
```

Compare it with the equivalent code in Java:

```
public class Hello
{
    public static void main(String[] args) {
        System.out.printf("Hello world!");
    }
}
```

Let’s see a code sample in C language. The following program reads a file (input.txt) and copies its contents into another file (output.txt):

```
#include <stdio.h>
int main(int argc, char **argv) {
    FILE *in, *out;
    int c;
    in = fopen("input.txt", "r");
    out = fopen("output.txt", "w");
    while ((c = fgetc(in)) != EOF) {
        fputc(c, out);
    }
    fclose(out);
    fclose(in);
}
```

The same program in Python is shorter and easier to read:

```
with open("input.txt") as input_file:
    with open("output.txt") as output_file:
        output_file.writelines(in)
```

Let’s see a Perl program that calculates the average of a series of numbers:

```

sub avg(@_) {
    $sum += $_ foreach @_;
    return $sum / @_ unless @_ == 0;
    return 0;
}
print avg((1..5))."\n";

```

The equivalent program in Python is:

```

def avg(data):
    if len(data)==0:
        return 0
    else:
        return sum(data)/len(data)
print(avg([1,2,3,4,5]))

```

The purpose of this Python program could be almost fully understood by just knowing English.

Python is designed to be a highly readable language.¹³ The use of English keywords, and the use of spaces to limit code blocks and its internal logic (indentation), contribute to this end. It's possible to write hard-to-read code in Python, but it requires a deliberate effort to obfuscate the code.¹⁴

Speed

Another criterion to consider when choosing a programming language is execution speed. In the early days of computer programming, computers were so slow that some differences due to language implementation were very significant. It could take a week for a program to be executed in an interpreted language, while the same code in a compiled language could be executed in a day. This performance difference between interpreted and compiled languages still has the same proportion, but it is less relevant. This is because a program that took a week to run, now takes less than ten seconds, while the compiled one takes about one second. Although the difference seems important (at least one order of magnitude), it is not so relevant if we consider the time it takes to develop it.

This does not mean that execution speed does not need to be considered. A 10X speed difference can be crucial in some high-performance computing operations. Sometimes a lot of improvements can be achieved by writing optimized code. If the code is written with speed optimization in mind, it is possible to obtain results quite

¹³Other languages are regarded as “write only,” since once written it is very difficult to understand it.

¹⁴A simple `print('Hello World')` program could be written, if you are so inclined, as `print(''.join([chr((L>=65 and L<=122) and (((L>=97) and (L-96) or (L-64))-1)+13)%26+((L>=97) and 97 or 65)) or L) for L in [ord(C) for C in 'Uryyb Jbeyq!']])` (<https://goo.gl/r5sm9j>).

similar to the ones that could be obtained in a compiled language. In the cases where the programmer is not satisfied with the speed obtained by Python, it is possible to link to an external library written in another language (like C or Fortran). This way, we can get the best of both worlds: the ease of Python programming with the speed of a compiled language.

1.5.3 How Is It Used?

Python has a wide range of applications. From cell phones to web servers, there are thousands of Python applications in the most diverse fields. There is Python code powering Wikipedia robots, helping design next generation special effects at Industrial Light & Magic,¹⁵ embedded in D-link modems and routers,¹⁶ and it is the scripting language of the OpenOffice suite¹⁷.

Some languages are strong in one niche (like PHP for web applications, Java for desktop programs), but Python can't be typecast easily.

With a single code base, Python desktop applications run with a native look and feel on multiple platforms. Well-known examples of this category include the **BitTorrent** p2p client/server, **Calibre**, an Ebook manager, **Sage Math** (a mathematics software system), the **Dropbox** client, and more.

As a language for building web applications, Python can be found in high traffic sites like Reddit, NationalGeographic, Instagram, and NASA. There are specialized software for building web sites (called webframeworks) in Python like **Django**, **Web2Py**, **Pyramid**, **Flask**, and **Bottle**.

From system administration to data analysis, Python provides a broad range of tools to this end:

- Generic Operating System Services (os, io, time, curses)
- File and Directory Access (os.path, glob, tempfile, shutil)
- Data Compression and Archiving (zipfile, gzip, bz2)
- Interprocess Communication and Networking (subprocess, socket, ssl)
- Internet (email, mimetools, rfc822, cgi, urllib)
- String Services (string, re, codecs, unicodedata)

Python is gaining momentum as the default computer language for the scientific community. There are several libraries oriented toward scientific users, such as **SciPy**¹⁸ and **Anaconda**.¹⁹ Both distributions integrate modules for linear algebra,

¹⁵<https://www.python.org/about/success/ilm/>

¹⁶<https://www.python.org/about/success/dlink/>

¹⁷<http://wiki.services.openoffice.org/wiki/Python>

¹⁸<https://www.scipy.org>

¹⁹<https://www.continuum.io/anaconda-overview>