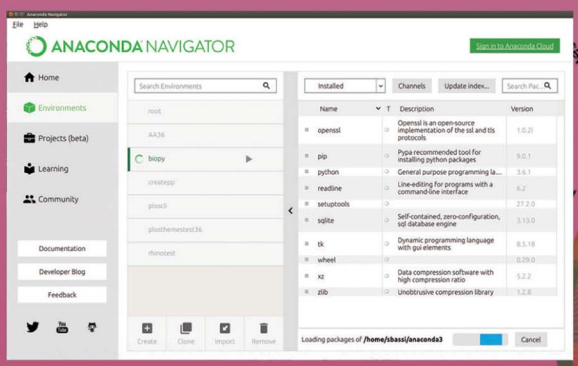
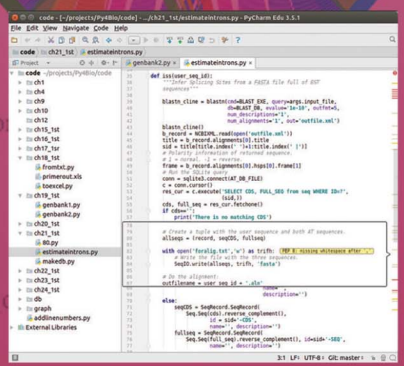
Chapman & Hall/CRC
Mathematical and Computational Biology Series

PYTHON FOR BIOINFORMATICS

SECOND EDITION





biopython

SEBASTIAN BASSI

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

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

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

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

Printed on acid-free paper
Version Date: 20170626

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

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

List of Tables

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 Azure[1]).

[1] See `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

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 LaTeX2ε 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.

I

Programming

Introduction

CONTENTS

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 maintained[1] 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:

[1] Python 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.

[2]The 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 it[3]), 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.

[3] In 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   H2O
1.0 mcL    EcoRI
```

[4]There 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_20, 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 we 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 visualization[6] and plotting, PDF generation, bioinformatics analysis,[7] image

[5] In Python the latter form is used.

[6] MatPlotLib (`http://matplotlib.org/`) and Bokeh `http://bokeh.pydata.org/en/latest/` are the most used.

[7] Biopython 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 MATLAB[11].

- 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

[8] Scikit-image paper: `http://peerj.com/articles/453`

[9] scikit-learn website: `http://scikit-learn.org/stable/`

[10] `https://wiki.python.org/moin/DatabaseProgramming`

[11] 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.

[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

[13] Other languages are regarded as "write only," since once written it is very difficult to understand it.

[14] A 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 suite[17].

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`