Python\_for\_Data\_Analysis\_Wes\_McKinney\_First\_release\_c01

APTER 1

Preliminaries

What Is This Book About?

This book is concerned with the nuts and bolts of manipulating, processing, cleaning,

and crunching data in Python. It is also a practical, modern introduction to scientific

computing in Python, tailored for data-intensive applications. This is a book about the

parts of the Python language and libraries you’ll need to effectively solve a broad set of

data analysis problems. This book is not an exposition on analytical methods using

Python as the implementation language.

When I say “data”, what am I referring to exactly? The primary focus is on structured

data, a deliberately vague term that encompasses many different common forms of

data, such as

• Multidimensional arrays (matrices)

• Tabular or spreadsheet-like data in which each column may be a different type

(string, numeric, date, or otherwise). This includes most kinds of data commonly

stored in relational databases or tab- or comma-delimited text files

• Multiple tables of data interrelated by key columns (what would be primary or

foreign keys for a SQL user)

• Evenly or unevenly spaced time series

This is by no means a complete list. Even though it may not always be obvious, a large

percentage of data sets can be transformed into a structured form that is more suitable

for analysis and modeling. If not, it may be possible to extract features from a data set

into a structured form. As an example, a collection of news articles could be processed

into a word frequency table which could then be used to perform sentiment analysis.

Most users of spreadsheet programs like Microsoft Excel, perhaps the most widely used

data analysis tool in the world, will not be strangers to these kinds of data.

1

www.it-ebooks.infoWhy Python for Data Analysis?

For many people (myself among them), the Python language is easy to fall in love with.

Since its first appearance in 1991, Python has become one of the most popular dynamic,

programming languages, along with Perl, Ruby, and others. Python and Ruby have

become especially popular in recent years for building websites using their numerous

web frameworks, like Rails (Ruby) and Django (Python). Such languages are often

called scripting languages as they can be used to write quick-and-dirty small programs,

or scripts. I don’t like the term “scripting language” as it carries a connotation that they

cannot be used for building mission-critical software. Among interpreted languages

Python is distinguished by its large and active scientific computing community. Adop-

tion of Python for scientific computing in both industry applications and academic

research has increased significantly since the early 2000s.

For data analysis and interactive, exploratory computing and data visualization, Python

will inevitably draw comparisons with the many other domain-specific open source

and commercial programming languages and tools in wide use, such as R, MATLAB,

SAS, Stata, and others. In recent years, Python’s improved library support (primarily

pandas) has made it a strong alternative for data manipulation tasks. Combined with

Python’s strength in general purpose programming, it is an excellent choice as a single

language for building data-centric applications.

Python as Glue

Part of Python’s success as a scientific computing platform is the ease of integrating C,

C++, and FORTRAN code. Most modern computing environments share a similar set

of legacy FORTRAN and C libraries for doing linear algebra, optimization, integration,

fast fourier transforms, and other such algorithms. The same story has held true for

many companies and national labs that have used Python to glue together 30 years’

worth of legacy software.

Most programs consist of small portions of code where most of the time is spent, with

large amounts of “glue code” that doesn’t run often. In many cases, the execution time

of the glue code is insignificant; effort is most fruitfully invested in optimizing the

computational bottlenecks, sometimes by moving the code to a lower-level language

like C.

In the last few years, the Cython project (http://cython.org) has become one of the

preferred ways of both creating fast compiled extensions for Python and also interfacing

with C and C++ code.

Solving the “Two-Language” Problem

In many organizations, it is common to research, prototype, and test new ideas using

a more domain-specific computing language like MATLAB or R then later port those

2 | Chapter 1: Preliminaries

www.it-ebooks.infoideas to be part of a larger production system written in, say, Java, C#, or C++. What

people are increasingly finding is that Python is a suitable language not only for doing

research and prototyping but also building the production systems, too. I believe that

more and more companies will go down this path as there are often significant organ-

izational benefits to having both scientists and technologists using the same set of pro-

grammatic tools.

Why Not Python?

While Python is an excellent environment for building computationally-intensive sci-

entific applications and building most kinds of general purpose systems, there are a

number of uses for which Python may be less suitable.

As Python is an interpreted programming language, in general most Python code will

run substantially slower than code written in a compiled language like Java or C++. As

programmer time is typically more valuable than CPU time, many are happy to make

this tradeoff. However, in an application with very low latency requirements (for ex-

ample, a high frequency trading system), the time spent programming in a lower-level,

lower-productivity language like C++ to achieve the maximum possible performance

might be time well spent.

Python is not an ideal language for highly concurrent, multithreaded applications, par-

ticularly applications with many CPU-bound threads. The reason for this is that it has

what is known as the global interpreter lock (GIL), a mechanism which prevents the

interpreter from executing more than one Python bytecode instruction at a time. The

technical reasons for why the GIL exists are beyond the scope of this book, but as of

this writing it does not seem likely that the GIL will disappear anytime soon. While it

is true that in many big data processing applications, a cluster of computers may be

required to process a data set in a reasonable amount of time, there are still situations

where a single-process, multithreaded system is desirable.

This is not to say that Python cannot execute truly multithreaded, parallel code; that

code just cannot be executed in a single Python process. As an example, the Cython

project features easy integration with OpenMP, a C framework for parallel computing,

in order to to parallelize loops and thus significantly speed up numerical algorithms.

Essential Python Libraries

For those who are less familiar with the scientific Python ecosystem and the libraries

used throughout the book, I present the following overview of each library.

Essential Python Libraries | 3

www.it-ebooks.infoNumPy

NumPy, short for Numerical Python, is the foundational package for scientific com-

puting in Python. The majority of this book will be based on NumPy and libraries built

on top of NumPy. It provides, among other things:

• A fast and efficient multidimensional array object ndarray

• Functions for performing element-wise computations with arrays or mathematical

operations between arrays

• Tools for reading and writing array-based data sets to disk

• Linear algebra operations, Fourier transform, and random number generation

• Tools for integrating connecting C, C++, and Fortran code to Python

Beyond the fast array-processing capabilities that NumPy adds to Python, one of its

primary purposes with regards to data analysis is as the primary container for data to

be passed between algorithms. For numerical data, NumPy arrays are a much more

efficient way of storing and manipulating data than the other built-in Python data

structures. Also, libraries written in a lower-level language, such as C or Fortran, can

operate on the data stored in a NumPy array without copying any data.

pandas

pandas provides rich data structures and functions designed to make working with

structured data fast, easy, and expressive. It is, as you will see, one of the critical in-

gredients enabling Python to be a powerful and productive data analysis environment.

The primary object in pandas that will be used in this book is the DataFrame, a two-

dimensional tabular, column-oriented data structure with both row and column labels:

>>> frame

total\_bill

1 16.99

2 10.34

3 21.01

4 23.68

5 24.59

6 25.29

7 8.77

8 26.88

9 15.04

10 14.78

tip

1.01

1.66

3.5

3.31

3.61

4.71

2

3.12

1.96

3.23

sex

Female

Male

Male

Male

Female

Male

Male

Male

Male

Male

smoker

No

No

No

No

No

No

No

No

No

No

day

Sun

Sun

Sun

Sun

Sun

Sun

Sun

Sun

Sun

Sun

time

Dinner

Dinner

Dinner

Dinner

Dinner

Dinner

Dinner

Dinner

Dinner

Dinner

size

2

3

3

2

4

4

2

4

2

2

pandas combines the high performance array-computing features of NumPy with the

flexible data manipulation capabilities of spreadsheets and relational databases (such

as SQL). It provides sophisticated indexing functionality to make it easy to reshape,

slice and dice, perform aggregations, and select subsets of data. pandas is the primary

tool that we will use in this book.

4 | Chapter 1: Preliminaries

www.it-ebooks.infoFor financial users, pandas features rich, high-performance time series functionality

and tools well-suited for working with financial data. In fact, I initially designed pandas

as an ideal tool for financial data analysis applications.

For users of the R language for statistical computing, the DataFrame name will be

familiar, as the object was named after the similar R data.frame object. They are not

the same, however; the functionality provided by data.frame in R is essentially a strict

subset of that provided by the pandas DataFrame. While this is a book about Python, I

will occasionally draw comparisons with R as it is one of the most widely-used open

source data analysis environments and will be familiar to many readers.

The pandas name itself is derived from panel data, an econometrics term for multidi-

mensional structured data sets, and Python data analysis itself.

matplotlib

matplotlib is the most popular Python library for producing plots and other 2D data

visualizations. It was originally created by John D. Hunter (JDH) and is now maintained

by a large team of developers. It is well-suited for creating plots suitable for publication.

It integrates well with IPython (see below), thus providing a comfortable interactive

environment for plotting and exploring data. The plots are also interactive; you can

zoom in on a section of the plot and pan around the plot using the toolbar in the plot

window.

IPython

IPython is the component in the standard scientific Python toolset that ties everything

together. It provides a robust and productive environment for interactive and explor-

atory computing. It is an enhanced Python shell designed to accelerate the writing,

testing, and debugging of Python code. It is particularly useful for interactively working

with data and visualizing data with matplotlib. IPython is usually involved with the

majority of my Python work, including running, debugging, and testing code.

Aside from the standard terminal-based IPython shell, the project also provides

• A Mathematica-like HTML notebook for connecting to IPython through a web

browser (more on this later).

• A Qt framework-based GUI console with inline plotting, multiline editing, and

syntax highlighting

• An infrastructure for interactive parallel and distributed computing

I will devote a chapter to IPython and how to get the most out of its features. I strongly

recommend using it while working through this book.

Essential Python Libraries | 5

www.it-ebooks.infoSciPy

SciPy is a collection of packages addressing a number of different standard problem

domains in scientific computing. Here is a sampling of the packages included:

• scipy.integrate: numerical integration routines and differential equation solvers

• scipy.linalg: linear algebra routines and matrix decompositions extending be-

yond those provided in numpy.linalg.

• scipy.optimize: function optimizers (minimizers) and root finding algorithms

• scipy.signal: signal processing tools

• scipy.sparse: sparse matrices and sparse linear system solvers

• scipy.special: wrapper around SPECFUN, a Fortran library implementing many

common mathematical functions, such as the gamma function

• scipy.stats: standard continuous and discrete probability distributions (density

functions, samplers, continuous distribution functions), various statistical tests,

and more descriptive statistics

• scipy.weave: tool for using inline C++ code to accelerate array computations

Together NumPy and SciPy form a reasonably complete computational replacement

for much of MATLAB along with some of its add-on toolboxes.

Installation and Setup

Since everyone uses Python for different applications, there is no single solution for

setting up Python and required add-on packages. Many readers will not have a complete

scientific Python environment suitable for following along with this book, so here I will

give detailed instructions to get set up on each operating system. I recommend using

one of the following base Python distributions:

• Enthought Python Distribution: a scientific-oriented Python distribution from En-

thought (http://www.enthought.com). This includes EPDFree, a free base scientific

distribution (with NumPy, SciPy, matplotlib, Chaco, and IPython) and EPD Full,

a comprehensive suite of more than 100 scientific packages across many domains.

EPD Full is free for academic use but has an annual subscription for non-academic

users.

• Python(x,y) (http://pythonxy.googlecode.com): A free scientific-oriented Python

distribution for Windows.

I will be using EPDFree for the installation guides, though you are welcome to take

another approach depending on your needs. At the time of this writing, EPD includes

Python 2.7, though this might change at some point in the future. After installing, you

will have the following packages installed and importable:

6 | Chapter 1: Preliminaries

www.it-ebooks.info• Scientific Python base: NumPy, SciPy, matplotlib, and IPython. These are all in-

cluded in EPDFree.

• IPython Notebook dependencies: tornado and pyzmq. These are included in EPD-

Free.

• pandas (version 0.8.2 or higher).

At some point while reading you may wish to install one or more of the following

packages: statsmodels, PyTables, PyQt (or equivalently, PySide), xlrd, lxml, basemap,

pymongo, and requests. These are used in various examples. Installing these optional

libraries is not necessary, and I would would suggest waiting until you need them. For

example, installing PyQt or PyTables from source on OS X or Linux can be rather

arduous. For now, it’s most important to get up and running with the bare minimum:

EPDFree and pandas.

For information on each Python package and links to binary installers or other help,

see the Python Package Index (PyPI, http://pypi.python.org). This is also an excellent

resource for finding new Python packages.

To avoid confusion and to keep things simple, I am avoiding discussion

of more complex environment management tools like pip and virtua-

lenv. There are many excellent guides available for these tools on the

Internet.

Some users may be interested in alternate Python implementations, such

as IronPython, Jython, or PyPy. To make use of the tools presented in

this book, it is (currently) necessary to use the standard C-based Python

interpreter, known as CPython.

Windows

To get started on Windows, download the EPDFree installer from http://www.en

thought.com, which should be an MSI installer named like epd\_free-7.3-1-win-

x86.msi. Run the installer and accept the default installation location C:\Python27. If

you had previously installed Python in this location, you may want to delete it manually

first (or using Add/Remove Programs).

Next, you need to verify that Python has been successfully added to the system path

and that there are no conflicts with any prior-installed Python versions. First, open a

command prompt by going to the Start Menu and starting the Command Prompt ap-

plication, also known as cmd.exe. Try starting the Python interpreter by typing

python. You should see a message that matches the version of EPDFree you installed:

C:\Users\Wes>python

Python 2.7.3 |EPD\_free 7.3-1 (32-bit)| (default, Apr 12 2012, 14:30:37) on win32

Type "credits", "demo" or "enthought" for more information.

>>>

Installation and Setup | 7

www.it-ebooks.infoIf you see a message for a different version of EPD or it doesn’t work at all, you will

need to clean up your Windows environment variables. On Windows 7 you can start

typing “environment variables” in the programs search field and select Edit environ

ment variables for your account. On Windows XP, you will have to go to Control

Panel > System > Advanced > Environment Variables. On the window that pops up,

you are looking for the Path variable. It needs to contain the following two directory

paths, separated by semicolons:

C:\Python27;C:\Python27\Scripts

If you installed other versions of Python, be sure to delete any other Python-related

directories from both the system and user Path variables. After making a path alterna-

tion, you have to restart the command prompt for the changes to take effect.

Once you can launch Python successfully from the command prompt, you need to

install pandas. The easiest way is to download the appropriate binary installer from

http://pypi.python.org/pypi/pandas. For EPDFree, this should be pandas-0.9.0.win32-

py2.7.exe. After you run this, let’s launch IPython and check that things are installed

correctly by importing pandas and making a simple matplotlib plot:

C:\Users\Wes>ipython --pylab

Python 2.7.3 |EPD\_free 7.3-1 (32-bit)|

Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.

?

-> Introduction and overview of IPython's features.

%quickref -> Quick reference.

help

-> Python's own help system.

object?

-> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: WXAgg].

For more information, type 'help(pylab)'.

In [1]: import pandas

In [2]: plot(arange(10))

If successful, there should be no error messages and a plot window will appear. You

can also check that the IPython HTML notebook can be successfully run by typing:

$ ipython notebook --pylab=inline

If you use the IPython notebook application on Windows and normally

use Internet Explorer, you will likely need to install and run Mozilla

Firefox or Google Chrome instead.

EPDFree on Windows contains only 32-bit executables. If you want or need a 64-bit

setup on Windows, using EPD Full is the most painless way to accomplish that. If you

would rather install from scratch and not pay for an EPD subscription, Christoph

Gohlke at the University of California, Irvine, publishes unofficial binary installers for

8 | Chapter 1: Preliminaries

www.it-ebooks.infoall of the book’s necessary packages (http://www.lfd.uci.edu/~gohlke/pythonlibs/) for 32-

and 64-bit Windows.

Apple OS X

To get started on OS X, you must first install Xcode, which includes Apple’s suite of

software development tools. The necessary component for our purposes is the gcc C

and C++ compiler suite. The Xcode installer can be found on the OS X install DVD

that came with your computer or downloaded from Apple directly.

Once you’ve installed Xcode, launch the terminal (Terminal.app) by navigating to

Applications > Utilities. Type gcc and press enter. You should hopefully see some-

thing like:

$ gcc

i686-apple-darwin10-gcc-4.2.1: no input files

Now you need to install EPDFree. Download the installer which should be a disk image

named something like epd\_free-7.3-1-macosx-i386.dmg. Double-click the .dmg file to

mount it, then double-click the .mpkg file inside to run the installer.

When the installer runs, it automatically appends the EPDFree executable path to

your .bash\_profile file. This is located at /Users/your\_uname/.bash\_profile:

# Setting PATH for EPD\_free-7.3-1

PATH="/Library/Frameworks/Python.framework/Versions/Current/bin:${PATH}"

export PATH

Should you encounter any problems in the following steps, you’ll want to inspect

your .bash\_profile and potentially add the above directory to your path.

Now, it’s time to install pandas. Execute this command in the terminal:

$ sudo easy\_install pandas

Searching for pandas

Reading http://pypi.python.org/simple/pandas/

Reading http://pandas.pydata.org

Reading http://pandas.sourceforge.net

Best match: pandas 0.9.0

Downloading http://pypi.python.org/packages/source/p/pandas/pandas-0.9.0.zip

Processing pandas-0.9.0.zip

Writing /tmp/easy\_install-H5mIX6/pandas-0.9.0/setup.cfg

Running pandas-0.9.0/setup.py -q bdist\_egg --dist-dir /tmp/easy\_install-H5mIX6/

pandas-0.9.0/egg-dist-tmp-RhLG0z

Adding pandas 0.9.0 to easy-install.pth file

Installed /Library/Frameworks/Python.framework/Versions/7.3/lib/python2.7/

site-packages/pandas-0.9.0-py2.7-macosx-10.5-i386.egg

Processing dependencies for pandas

Finished processing dependencies for pandas

To verify everything is working, launch IPython in Pylab mode and test importing pan-

das then making a plot interactively:

Installation and Setup | 9

www.it-ebooks.info$ ipython --pylab

22:29 ~/VirtualBox VMs/WindowsXP $ ipython

Python 2.7.3 |EPD\_free 7.3-1 (32-bit)| (default, Apr 12 2012, 11:28:34)

Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.

?

-> Introduction and overview of IPython's features.

%quickref -> Quick reference.

help

-> Python's own help system.

object?

-> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: WXAgg].

For more information, type 'help(pylab)'.

In [1]: import pandas

In [2]: plot(arange(10))

If this succeeds, a plot window with a straight line should pop up.

GNU/Linux

Some, but not all, Linux distributions include sufficiently up-to-date

versions of all the required Python packages and can be installed using

the built-in package management tool like apt. I detail setup using EPD-

Free as it's easily reproducible across distributions.

Linux details will vary a bit depending on your Linux flavor, but here I give details for

Debian-based GNU/Linux systems like Ubuntu and Mint. Setup is similar to OS X with

the exception of how EPDFree is installed. The installer is a shell script that must be

executed in the terminal. Depending on whether you have a 32-bit or 64-bit system,

you will either need to install the x86 (32-bit) or x86\_64 (64-bit) installer. You will then

have a file named something similar to epd\_free-7.3-1-rh5-x86\_64.sh. To install it,

execute this script with bash:

$ bash epd\_free-7.3-1-rh5-x86\_64.sh

After accepting the license, you will be presented with a choice of where to put the

EPDFree files. I recommend installing the files in your home directory, say /home/wesm/

epd (substituting your own username for wesm).

Once the installer has finished, you need to add EPDFree’s bin directory to your

$PATH variable. If you are using the bash shell (the default in Ubuntu, for example), this

means adding the following path addition in your .bashrc:

export PATH=/home/wesm/epd/bin:$PATH

Obviously, substitute the installation directory you used for /home/wesm/epd/. After

doing this you can either start a new terminal process or execute your .bashrc again

with source ~/.bashrc.

10 | Chapter 1: Preliminaries

www.it-ebooks.infoYou need a C compiler such as gcc to move forward; many Linux distributions include

gcc, but others may not. On Debian systems, you can install gcc by executing:

sudo apt-get install gcc

If you type gcc on the command line it should say something like:

$ gcc

gcc: no input files

Now, time to install pandas:

$ easy\_install pandas

If you installed EPDFree as root, you may need to add sudo to the command and enter

the sudo or root password. To verify things are working, perform the same checks as

in the OS X section.

Python 2 and Python 3

The Python community is currently undergoing a drawn-out transition from the Python

2 series of interpreters to the Python 3 series. Until the appearance of Python 3.0, all

Python code was backwards compatible. The community decided that in order to move

the language forward, certain backwards incompatible changes were necessary.

I am writing this book with Python 2.7 as its basis, as the majority of the scientific

Python community has not yet transitioned to Python 3. The good news is that, with

a few exceptions, you should have no trouble following along with the book if you

happen to be using Python 3.2.

Integrated Development Environments (IDEs)

When asked about my standard development environment, I almost always say “IPy-

thon plus a text editor”. I typically write a program and iteratively test and debug each

piece of it in IPython. It is also useful to be able to play around with data interactively

and visually verify that a particular set of data manipulations are doing the right thing.

Libraries like pandas and NumPy are designed to be easy-to-use in the shell.

However, some will still prefer to work in an IDE instead of a text editor. They do

provide many nice “code intelligence” features like completion or quickly pulling up

the documentation associated with functions and classes. Here are some that you can

explore:

• Eclipse with PyDev Plugin

• Python Tools for Visual Studio (for Windows users)

• PyCharm

• Spyder

• Komodo IDE

Installation and Setup | 11

www.it-ebooks.infoCommunity and Conferences

Outside of an Internet search, the scientific Python mailing lists are generally helpful

and responsive to questions. Some ones to take a look at are:

• pydata: a Google Group list for questions related to Python for data analysis and

pandas

• pystatsmodels: for statsmodels or pandas-related questions

• numpy-discussion: for NumPy-related questions

• scipy-user: for general SciPy or scientific Python questions

I deliberately did not post URLs for these in case they change. They can be easily located

via Internet search.

Each year many conferences are held all over the world for Python programmers. PyCon

and EuroPython are the two main general Python conferences in the United States and

Europe, respectively. SciPy and EuroSciPy are scientific-oriented Python conferences

where you will likely find many “birds of a feather” if you become more involved with

using Python for data analysis after reading this book.

Navigating This Book

If you have never programmed in Python before, you may actually want to start at the

end of the book, where I have placed a condensed tutorial on Python syntax, language

features, and built-in data structures like tuples, lists, and dicts. These things are con-

sidered prerequisite knowledge for the remainder of the book.

The book starts by introducing you to the IPython environment. Next, I give a short

introduction to the key features of NumPy, leaving more advanced NumPy use for

another chapter at the end of the book. Then, I introduce pandas and devote the rest

of the book to data analysis topics applying pandas, NumPy, and matplotlib (for vis-

ualization). I have structured the material in the most incremental way possible, though

there is occasionally some minor cross-over between chapters.

Data files and related material for each chapter are hosted as a git repository on GitHub:

http://github.com/pydata/pydata-book

I encourage you to download the data and use it to replicate the book’s code examples

and experiment with the tools presented in each chapter. I will happily accept contri-

butions, scripts, IPython notebooks, or any other materials you wish to contribute to

the book's repository for all to enjoy.

12 | Chapter 1: Preliminaries

www.it-ebooks.infoCode Examples

Most of the code examples in the book are shown with input and output as it would

appear executed in the IPython shell.

In [5]: code

Out[5]: output

At times, for clarity, multiple code examples will be shown side by side. These should

be read left to right and executed separately.

In [5]: code

Out[5]: output

In [6]: code2

Out[6]: output2

Data for Examples

Data sets for the examples in each chapter are hosted in a repository on GitHub: http:

//github.com/pydata/pydata-book. You can download this data either by using the git

revision control command-line program or by downloading a zip file of the repository

from the website.

I have made every effort to ensure that it contains everything necessary to reproduce

the examples, but I may have made some mistakes or omissions. If so, please send me

an e-mail: wesmckinn@gmail.com.

Import Conventions

The Python community has adopted a number of naming conventions for commonly-

used modules:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

This means that when you see np.arange, this is a reference to the arange function in

NumPy. This is done as it’s considered bad practice in Python software development

to import everything (from numpy import \*) from a large package like NumPy.

Jargon

I’ll use some terms common both to programming and data science that you may not

be familiar with. Thus, here are some brief definitions:

Munge/Munging/Wrangling

Describes the overall process of manipulating unstructured and/or messy data into

a structured or clean form. The word has snuck its way into the jargon of many

modern day data hackers. Munge rhymes with “lunge”.

Navigating This Book | 13

www.it-ebooks.infoPseudocode

A description of an algorithm or process that takes a code-like form while likely

not being actual valid source code.

Syntactic sugar

Programming syntax which does not add new features, but makes something more

convenient or easier to type.

Acknowledgements

It would have been difficult for me to write this book without the support of a large

number of people.

On the O’Reilly staff, I’m very grateful for my editors Meghan Blanchette and Julie

Steele who guided me through the process. Mike Loukides also worked with me in the

proposal stages and helped make the book a reality.

I received a wealth of technical review from a large cast of characters. In particular,

Martin Blais and Hugh White were incredibly helpful in improving the book’s exam-

ples, clarity, and organization from cover to cover. James Long, Drew Conway, Fer-

nando Pérez, Brian Granger, Thomas Kluyver, Adam Klein, Josh Klein, Chang She, and

Stéfan van der Walt each reviewed one or more chapters, providing pointed feedback

from many different perspectives.

I got many great ideas for examples and data sets from friends and colleagues in the

data community, among them: Mike Dewar, Jeff Hammerbacher, James Johndrow,

Kristian Lum, Adam Klein, Hilary Mason, Chang She, and Ashley Williams.

I am of course indebted to the many leaders in the open source scientific Python com-

munity who’ve built the foundation for my development work and gave encouragement

while I was writing this book: the IPython core team (Fernando Pérez, Brian Granger,

Min Ragan-Kelly, Thomas Kluyver, and others), John Hunter, Skipper Seabold, Travis

Oliphant, Peter Wang, Eric Jones, Robert Kern, Josef Perktold, Francesc Alted, Chris

Fonnesbeck, and too many others to mention. Several other people provided a great

deal of support, ideas, and encouragement along the way: Drew Conway, Sean Taylor,

Giuseppe Paleologo, Jared Lander, David Epstein, John Krowas, Joshua Bloom, Den

Pilsworth, John Myles-White, and many others I’ve forgotten.

I’d also like to thank a number of people from my formative years. First, my former

AQR colleagues who’ve cheered me on in my pandas work over the years: Alex Reyf-

man, Michael Wong, Tim Sargen, Oktay Kurbanov, Matthew Tschantz, Roni Israelov,

Michael Katz, Chris Uga, Prasad Ramanan, Ted Square, and Hoon Kim. Lastly, my

academic advisors Haynes Miller (MIT) and Mike West (Duke).

On the personal side, Casey Dinkin provided invaluable day-to-day support during the

writing process, tolerating my highs and lows as I hacked together the final draft on

14 | Chapter 1: Preliminaries

www.it-ebooks.infotop of an already overcommitted schedule. Lastly, my parents, Bill and Kim, taught me

to always follow my dreams and to never settle for less.

Acknowledgements | 15

www.it-ebooks.infowww.it-ebooks.infoCHAPTER 2

Introdu