

深入浅出Python (影印版)

# Head First Python



Load important  
programming concepts  
directly into your brain



Model data  
with lists,  
sets, and  
dictionaries

Preserve  
your data  
in a pickle



Hook up with  
JSON, Android,  
and App Engine



Move your custom  
app to the Web



Share your code with  
the world on PyPI

O'REILLY® 東南大學出版社

Paul Barry 著

深入浅出 Python (影印版)

Head First Python

Wouldn't it be dreamy if there  
were a Python book that didn't  
make you wish you were anywhere  
other than stuck in front of your  
computer writing code? I guess it's  
just a fantasy...



Paul Barry

**O'REILLY®**

Beijing • Cambridge • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

## 图书在版编目 (CIP) 数据

深入浅出Python: 英文 / (英) 巴里 (Barry, P.) 著. —  
影印本. —南京: 东南大学出版社, 2011.5  
书名原文: Head First Python  
ISBN 978-7-5641-2675-9

I. ①深… II. ①巴… III. ①软件工具—程序设计  
—英文 IV. ①TP311.56

中国版本图书馆 CIP 数据核字 (2011) 第 042748 号

江苏省版权局著作权合同登记

图字: 10-2010-439 号

©2010 by O'Reilly Media, Inc.

Reprint of the English Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2011. Authorized reprint of the original English edition, 2010 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2010。

英文影印版由东南大学出版社出版 2011。此影印版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有, 未得书面许可, 本书的任何部分和全部不得以任何形式重制。

## 深入浅出 Python (影印版)

---

出版发行: 东南大学出版社

地 址: 南京四牌楼 2 号 邮编: 210096

出 版 人: 江建中

网 址: <http://www.seupress.com>

电子邮件: [press@seupress.com](mailto:press@seupress.com)

印 刷: 扬中市印刷有限公司

开 本: 787 毫米 × 980 毫米 12 开本

印 张: 41.5 印张

字 数: 694 千字

版 次: 2011 年 5 月第 1 版

印 次: 2011 年 5 月第 1 次印刷

书 号: ISBN 978-7-5641-2675-9

定 价: 88.00 元 (册)

---

本社图书若有印装质量问题, 请直接与读者服务部联系。电话 (传真): 025-83792328

# Table of Contents (Summary)

	Intro	xxiii
1	Meet Python: <i>Everyone Loves Lists</i>	1
2	Sharing Your Code: <i>Modules of Functions</i>	33
3	Files and Exceptions: <i>Dealing with Errors</i>	73
4	Persistence: <i>Saving Data to Files</i>	105
5	Comprehending Data: <i>Work That Data!</i>	139
6	Custom Data Objects: <i>Bundling Code with Data</i>	173
7	Web Development: <i>Putting It All Together</i>	213
8	Mobile App Development: <i>Small Devices</i>	255
9	Manage Your Data: <i>Handling Input</i>	293
10	Scaling Your Webapp: <i>Getting Real</i>	351
11	Dealing with Complexity: <i>Data Wrangling</i>	397
i	Leftovers: <i>The Top Ten Things (We Didn't Cover)</i>	435

# Table of Contents (the real thing)

## Intro

**Your brain on Python.** Here you are trying to learn something, while here your brain is doing you a favor by making sure the learning doesn't stick. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how do you trick your brain into thinking that your life depends on knowing Python?

Who is this book for?	xxiv
We know what you're thinking	xxv
Metacognition	xxvii
Bend your brain into submission	xxix
Read me	xxx
The technical review team	xxxii
Acknowledgments	xxxiii

meet python

## Everyone loves lists

1

### You're asking one question: "What makes Python different?"

The short answer is: *lots of things*. The longer answer starts by stating that there's lots that's familiar, too. Python is a lot like any other *general-purpose* programming language, with **statements**, **expressions**, **operators**, **functions**, **modules**, **methods**, and **classes**. All the *usual stuff*, really. And then there's the other stuff Python provides that makes the programmer's life—*your* life—that little bit easier. You'll start your tour of Python by learning about **lists**. But, before getting to that, there's another important question that needs answering...

What's to like about Python?	2
Install Python 3	3
Use IDLE to help learn Python	4
Work effectively with IDLE	5
Deal with complex data	6
Create simple Python lists	7
Lists are like arrays	9
Add more data to your list	11
Work with your list data	15
For loops work with lists of any size	16
Store lists within lists	18
Check a list for a list	20
Complex data is hard to process	23
Handle many levels of nested lists	24
Don't repeat code; create a function	28
Create a function in Python	29
Recursion to the rescue!	31
Your Python Toolbox	32

The Holy Grail, 1975, Terry Jones & Terry Gilliam, 91 mins

Graham Chapman

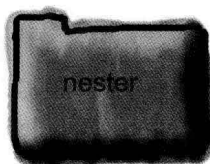
Michael Palin, John Cleese, Terry Gilliam, Eric Idle & Terry Jones

## sharing your code

**Modules of functions****2****Reusable code is great, but a shareable module is better.**

By sharing your code as a Python module, you open up your code to the entire Python community...and it's always good to share, isn't it? In this chapter, you'll learn how to create, install, and distribute your own shareable modules. You'll then load your module onto Python's software sharing site on the Web, so that *everyone* can benefit from your work. Along the way, you'll pick up a few new tricks relating to Python's functions, too.

It's too good not to share	34
Turn your function into a module	35
Modules are everywhere	36
Comment your code	37
Prepare your distribution	40
Build your distribution	41
A quick review of your distribution	42
Import a module to use it	43
Python's modules implement namespaces	45
Register with the PyPI website	47
Upload your code to PyPI	48
Welcome to the PyPI community	49
Control behavior with an extra argument	52
Before you write new code, think BIF	53
Python tries its best to run your code	57
Trace your code	58
Work out what's wrong	59
Update PyPI with your new code	60
You've changed your API	62
Use optional arguments	63
Your module supports both APIs	65
Your API is still not right	66
Your module's reputation is restored	70
Your Python Toolbox	71



```

├── nester.py
└── setup.py

```

## files and exceptions

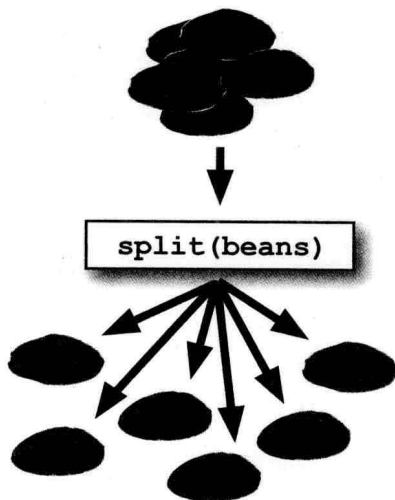
### Dealing with errors

# 3

#### It's simply not enough to process your list data in your code.

You need to be able to get your data *into* your programs with ease, too. It's no surprise then that Python makes reading data from **files** easy. Which is great, until you consider what can go *wrong* when interacting with data *external* to your programs... and there are lots of things waiting to trip you up! When bad stuff happens, you need a strategy for getting out of trouble, and one such strategy is to deal with any exceptional situations using Python's **exception handling** mechanism showcased in this chapter.

Data is external to your program	74
It's all lines of text	75
Take a closer look at the data	77
Know your data	79
Know your methods and ask for help	80
Know your data (better)	82
Two very different approaches	83
Add extra logic	84
Handle exceptions	88
Try first, then recover	89
Identify the code to protect	91
Take a pass on the error	93
What about other errors?	96
Add more error-checking code...	97
...Or add another level of exception handling	98
So, which approach is best?	99
You're done...except for one small thing	101
Be specific with your exceptions	102
Your Python Toolbox	103



persistence

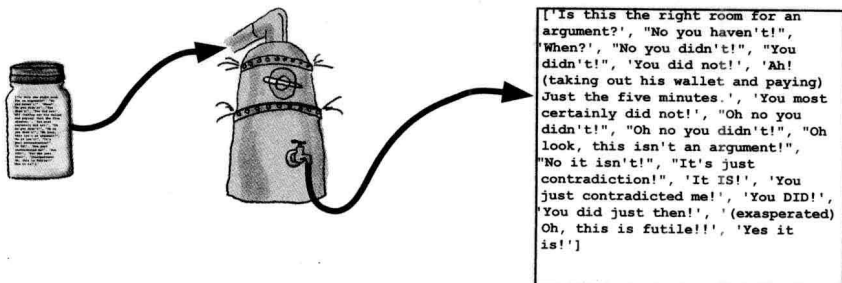
**Saving data to files**

4

**It is truly great to be able to process your file-based data.**

But what happens to your data when you're done? Of course, it's best to save your data to a disk file, which allows you to use it again at some later date and time. Taking your memory-based data and storing it to disk is what **persistence** is all about. Python supports all the usual tools for writing to files and also provides some cool facilities for *efficiently* storing Python data.

Programs produce data	106
Open your file in write mode	110
Files are left open after an exception!	114
Extend try with finally	115
Knowing the type of error is not enough	117
Use with to work with files	120
Default formats are unsuitable for files	124
Why not modify print_lol()?	126
Pickle your data	132
Save with dump and restore with load	133
Generic file I/O with pickle is the way to go!	137
Your Python Toolbox	138





## comprehending data

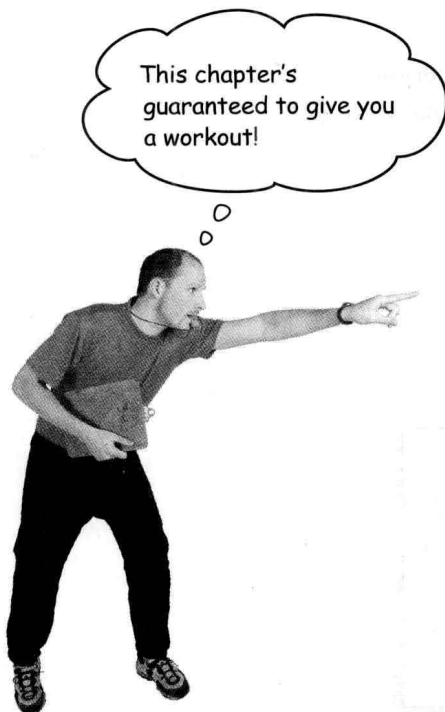
### Work that data!

# 5

#### Data comes in all shapes and sizes, formats and encodings.

To work effectively with your data, you often have to manipulate and transform it into a common format to allow for efficient processing, sorting, and storage. In this chapter, you'll explore Python goodies that help you work your data up into a sweat, allowing you to achieve data-munging greatness.

Coach Kelly needs your help	140
Sort in one of two ways	144
The trouble with time	148
Comprehending lists	155
Iterate to remove duplicates	161
Remove duplicates with sets	166
Your Python Toolbox	172

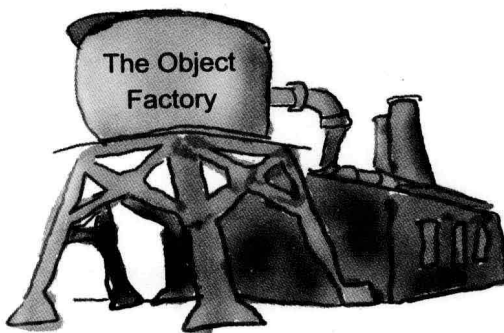


custom data objects

**Bundling code with data****6****It's important to match your data structure choice to your data.**

And that choice can make a big difference to the complexity of your code. In Python, although really useful, lists and sets aren't the only game in town. The Python **dictionary** lets you organize your data for speedy lookup by *associating your data with names*, not numbers. And when Python's built-in data structures don't quite cut it, the Python **class** statement lets you define your own. This chapter shows you how.

Coach Kelly is back (with a new file format)	174
Use a dictionary to associate data	178
Bundle your code and its data in a class	189
Define a class	190
Use class to define classes	191
The importance of self	192
Every method's first argument is self	193
Inherit from Python's built-in list	204
Coach Kelly is impressed	211
Your Python Toolbox	212



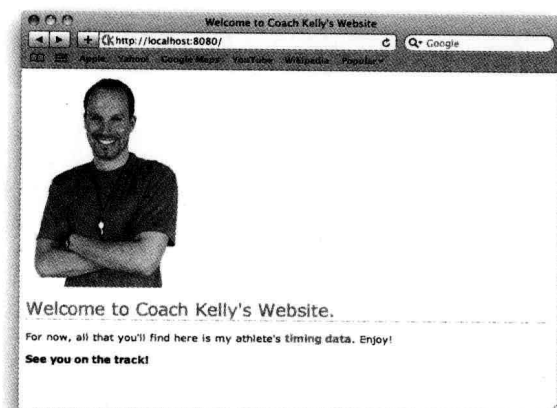
## web development

## 7

**Putting it all together****Sooner or later, you'll want to share your app with lots of people.**

You have many options for doing this. Pop your code on PyPI, send out lots of emails, put your code on a CD or USB, or simply install your app manually on the computers of those people who need it. Sounds like a lot of work...not to mention boring. Also, what happens when you produce the next best version of your code? What happens then? How do you manage the update? Let's face it: it's such a pain that you'll think up really creative excuses not to. Luckily, you don't have to do any of this: just create a webapp instead. And, as this chapter demonstrates, using Python for web development is a breeze.

It's good to share	214
You can put your program on the Web	215
What does your webapp need to do?	218
Design your webapp with MVC	221
Model your data	222
View your interface	226
Control your code	234
CGI lets your web server run programs	235
Display the list of athletes	236
The dreaded 404 error!	242
Create another CGI script	244
Enable CGI tracking to help with errors	248
A small change can make all the difference	250
Your webapp's a hit!	252
Your Python Toolbox	253



## mobile app development

**Small devices****8****Putting your data on the Web opens up all types of possibilities.**

Not only can anyone from anywhere interact with your webapp, but they are increasingly doing so from a collection of diverse computing devices: PCs, laptops, tablets, palmtops, and even mobile phones. And it's not just humans interacting with your webapp that you have to support and worry about: *bots* are small programs that can automate web interactions and typically want your data, not your human-friendly HTML. In this chapter, you exploit Python on Coach Kelly's mobile phone to write an app that interacts with your webapp's data.

The world is getting smaller	256
Coach Kelly is on Android	257
Don't worry about Python 2	259
Set up your development environment	260
Configure the SDK and emulator	261
Install and configure Android Scripting	262
Add Python to your SL4A installation	263
Test Python on Android	264
Define your app's requirements	266
The SL4A Android API	274
Select from a list on Android	278
The athlete's data CGI script	281
The data appears to have changed type	284
JSON can't handle your custom datatypes	285
Run your app on a real phone	288
Configure AndFTP	289
The coach is thrilled with his app	290
Your Python Toolbox	291



## manage your data

**Handling input**

## 9

**The Web and your phone are not just great ways to display data.**

They are also great tools to for accepting input from your users. Of course, once your webapp accepts data, it needs to put it somewhere, and the choices you make when deciding what and where this “somewhere” is are often the difference between a webapp that’s easy to grow and extend and one that isn’t. In this chapter, you’ll extend your webapp to accept data from the Web (via a browser or from an Android phone), as well as look at and enhance your back-end data-management services.

Your athlete times app has gone national	294
Use a form or dialog to accept input	295
Create an HTML form template	296
The data is delivered to your CGI script	300
Ask for input on your Android phone	304
It’s time to update your server data	308
Avoid race conditions	309
You need a better data storage mechanism	310
Use a database management system	312
Python includes SQLite	313
Exploit Python’s database API	314
The database API as Python code	315
A little database design goes a long way	316
Define your database schema	317
What does the data look like?	318
Transfer the data from your pickle to SQLite	321
What ID is assigned to which athlete?	322
Insert your timing data	323
SQLite data management tools	326
Integrate SQLite with your existing webapp	327
You still need the list of names	332
Get an athlete’s details based on ID	333
You need to amend your Android app, too	342
Update your SQLite-based athlete data	348
The NUAC is over the moon!	349
Your Python Toolbox	350



## scaling your webapp

## Getting real

## 10

## The Web is a great place to host your app...until things get real.

Sooner or later, you'll hit the jackpot and your webapp will be *wildly successful*. When that happens, your webapp goes from a handful of hits a day to thousands, possibly ten of thousands, *or even more*. Will you be ready? Will your web server handle the **load**? How will you know? What will it **cost**? *Who will pay*? Can your data model scale to millions upon millions of data items without *slowing to a crawl*? Getting a webapp up and running is easy with Python and now, thanks to Google App Engine, **scaling** a Python webapp is achievable, too.

There are whale sightings everywhere	352
The HFWWG needs to automate	353
Build your webapp with Google App Engine	354
Download and install App Engine	355
Make sure App Engine is working	356
App Engine uses the MVC pattern	359
Model your data with App Engine	360
What good is a model without a view?	363
Use templates in App Engine	364
Django's form validation framework	368
Check your form	369
Controlling your App Engine webapp	370
Restrict input by providing options	376
Meet the "blank screen of death"	378
Process the POST within your webapp	379
Put your data in the datastore	380
Don't break the "robustness principle"	384
Accept almost any date and time	385
It looks like you're not quite done yet	388
Sometimes, the tiniest change can make all the difference...	389
Capture your user's Google ID, too	390
Deploy your webapp to Google's cloud	391
Your HFWWG webapp is deployed!	394
Your Python Toolbox	395



## 11

## dealing with complexity

## Data wrangling

It's great when you can apply Python to a specific domain area.

Whether it's *web development*, *database management*, or *mobile apps*, Python helps you **get the job done** by *not* getting in the way of you coding your solution. And then there's the other types of problems: the ones you can't categorize or attach to a domain. Problems that are in themselves so *unique* you have to look at them in a different, highly specific way. Creating **bespoke** software solutions to these type of problems is an area where Python *excels*. In this, your final chapter, you'll *stretch* your Python skills to the limit and solve problems along the way.

What's a good time goal for the next race?	398
So...what's the problem?	400
Start with the data	401
Store each time as a dictionary	407
Dissect the prediction code	409
Get input from your user	413
Getting input raises an issue...	414
Search for the closest match	416
The trouble is with time	418
The time-to-seconds-to-time module	419
The trouble is still with time...	422
Port to Android	424
Your Android app is a bunch of dialogs	425
Put your app together...	429
Your app's a wrap!	431
Your Python Toolbox	432

File Edit View Insert Format Form Tools Help										
Formula: V02										
	A	B	C	D	E	F	G	H	I	
1	V02	84.8	82.9	81.1	79.3	77.5	75.8	74.2	72.5	
2	2mi	8:00	8:10	8:21	8:33	8:44	8:56	9:08	9:20	
3	5k	12:49	13:06	13:24	13:42	14:00	14:19	14:38	14:58	
4	5mi	21:19	21:48	22:17	22:47	23:18	23:50	24:22	24:55	
5	10k	26:54	27:30	28:08	28:45	29:24	30:04	30:45	31:28	
6	15k	41:31	42:27	43:24	44:23	45:23	46:24	47:27	48:31	
7	10mi	44:46	45:46	46:48	47:51	48:56	50:02	51:09	52:18	
8	20k	56:29	57:45	59:03	1:00:23	1:01:45	1:03:08	1:04:33	1:06:00	
9	13.1mi	59:49	1:01:09	1:02:32	1:03:56	1:05:23	1:06:51	1:08:21	1:09:53	
10	25k	1:11:43	1:13:20	1:14:59	1:16:40	1:18:24	1:20:10	1:21:58	1:23:49	
11	30k	1:27:10	1:19:08	1:31:08	1:33:11	1:35:17	1:37:26	1:39:37	1:41:52	
12	Marathon	2:05:34	2:08:24	2:11:17	2:14:15	2:17:16	2:20:21	2:23:31	2:26:44	

# 1 meet python

## ★ *Everyone loves lists* ★

Yes, yes...we have lots of Pythons in stock... I'll just make a quick list.



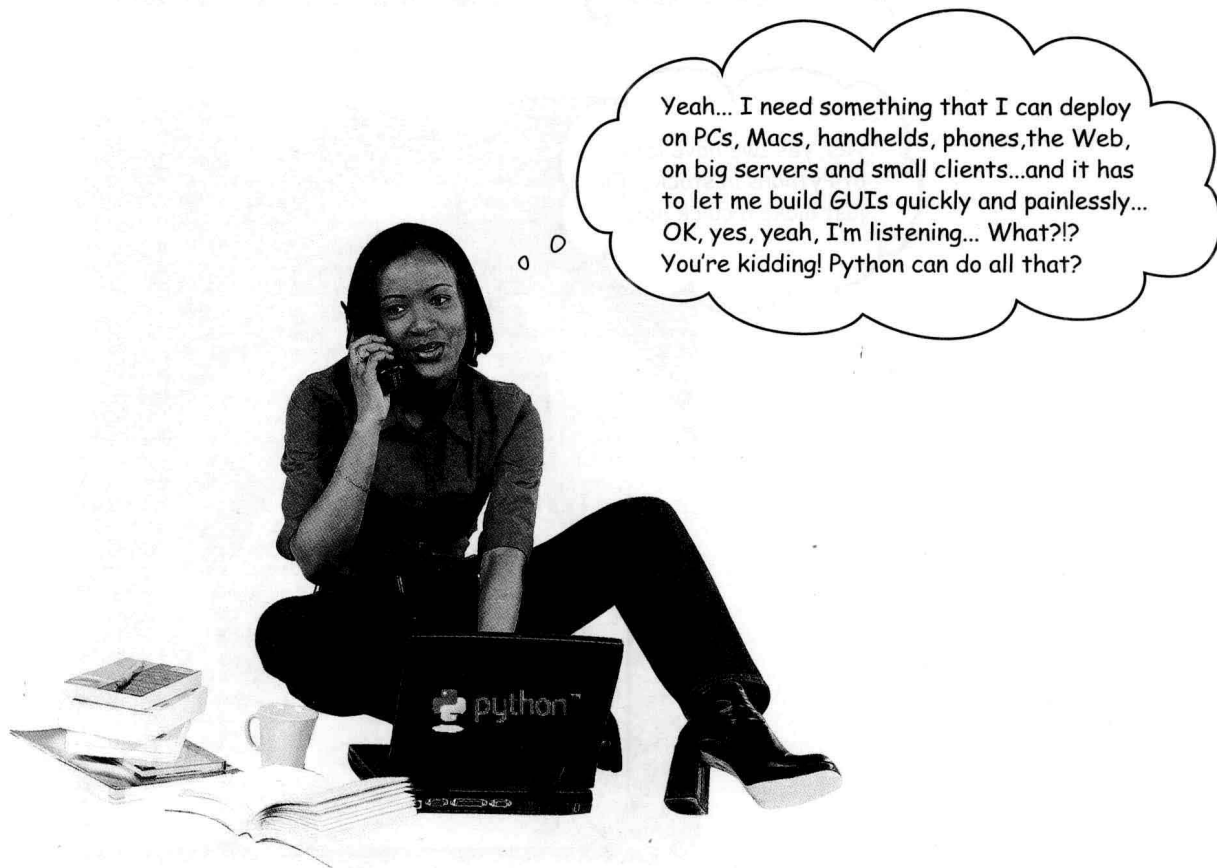
### **You're asking one question: "What makes Python different?"**

The short answer is: *lots of things*. The longer answer starts by stating that there's lots that's familiar, too. Python is a lot like any other *general-purpose* programming language, with **statements**, **expressions**, **operators**, **functions**, **modules**, **methods**, and **classes**. All the *usual stuff*, really. And then there's the other stuff Python provides that makes the programmer's life—*your* life—that little bit easier. You'll start your tour of Python by learning about **lists**. But, before getting to that, there's another important question that needs answering...



## What's to like about Python?

Lots. Rather than tell you, this book's goal is to *show* you the greatness that is Python.



Before diving head first into Python, let's get a bit of housekeeping out of the way.

To work with and execute the Python code in this book, you need a copy of the **Python 3 interpreter** on your computer. Like a lot of things to do with Python, it's not difficult to install the interpreter. Assuming, of course, it's not already there...