

Table of Contents

99	A Performance Experiment	99
99	Hash Tables as Dictionaries	99
32	Practical Consequences of How dict Works	32
33	How Sets Work—Practical Consequences	33
34	Chapter Summary	34
35	Further Reading	35
36		36
36	Text versus Bytes	101
37	Character Issues	101
38	Byte Essentials	101
39	Structs and Memory Views	101
39	Basic Encoders/Decoders	101
41	Understanding Encode/Decode Problems	101
42	Coping with UnicodeEncodeError	101
	Preface	xv
	Part I. Prologue	
	1. The Python Data Model	3
	A Pythonic Card Deck	4
	How Special Methods Are Used	8
	Emulating Numeric Types	9
	String Representation	11
	Arithmetic Operators	12
	Boolean Value of a Custom Type	12
	Overview of Special Methods	13
	Why len Is Not a Method	14
	Chapter Summary	15
	Further Reading	15
	Part II. Data Structures	
	2. An Array of Sequences	21
	Overview of Built-In Sequences	22
	List Comprehensions and Generator Expressions	23
	List Comprehensions and Readability	23
	Listcomps Versus map and filter	25
	Cartesian Products	26
	Generator Expressions	27
	Tuples Are Not Just Immutable Lists	28

Tuples as Records	29
Tuple Unpacking	30
Nested Tuple Unpacking	32
Named Tuples	33
Tuples as Immutable Lists	34
Slicing	35
Why Slices and Range Exclude the Last Item	36
Slice Objects	36
Multidimensional Slicing and Ellipsis	37
Assigning to Slices	38
Using + and * with Sequences	39
Building Lists of Lists	39
Augmented Assignment with Sequences	41
A += Assignment Puzzle	42
list.sort and the sorted Built-In Function	45
Managing Ordered Sequences with bisect	47
Searching with bisect	47
Inserting with bisect.insort	50
When a List Is Not the Answer	51
Arrays	51
Memory Views	54
NumPy and SciPy	55
Deque and Other Queues	58
Chapter Summary	61
Further Reading	62
3. Dictionaries and Sets	67
Generic Mapping Types	68
dict Comprehensions	70
Overview of Common Mapping Methods	71
Handling Missing Keys with setdefault	72
Mappings with Flexible Key Lookup	74
defaultdict: Another Take on Missing Keys	75
The __missing__ Method	76
Variations of dict	79
Subclassing UserDict	80
Immutable Mappings	82
Set Theory	83
set Literals	85
Set Comprehensions	86
Set Operations	87
dict and set Under the Hood	90

A Performance Experiment	90
Hash Tables in Dictionaries	92
Practical Consequences of How dict Works	95
How Sets Work—Practical Consequences	98
Chapter Summary	98
Further Reading	99
4. Text versus Bytes.....	101
Character Issues	102
Byte Essentials	103
Structs and Memory Views	106
Basic Encoders/Decoders	107
Understanding Encode/Decode Problems	109
Coping with UnicodeEncodeError	109
Coping with UnicodeDecodeError	110
SyntaxError When Loading Modules with Unexpected Encoding	112
How to Discover the Encoding of a Byte Sequence	113
BOM: A Useful Gremlin	114
Handling Text Files	115
Encoding Defaults: A Madhouse	118
Normalizing Unicode for Saner Comparisons	121
Case Folding	124
Utility Functions for Normalized Text Matching	125
Extreme “Normalization”: Taking Out Diacritics	126
Sorting Unicode Text	129
Sorting with the Unicode Collation Algorithm	131
The Unicode Database	132
Dual-Mode str and bytes APIs	134
str Versus bytes in Regular Expressions	134
str Versus bytes on os Functions	135
Chapter Summary	138
Further Reading	139

Part III. Functions as Objects

5. First-Class Functions.....	145
Treating a Function Like an Object	146
Higher-Order Functions	147
Modern Replacements for map, filter, and reduce	148
Anonymous Functions	150
The Seven Flavors of Callable Objects	151

90	User-Defined Callable Types	152
92	Function Introspection	153
92	From Positional to Keyword-Only Parameters	155
98	Retrieving Information About Parameters	156
98	Function Annotations	161
99	Packages for Functional Programming	163
	The operator Module	163
101	Freezing Arguments with <code>functools.partial</code>	167
102	Chapter Summary	169
103	Further Reading	169
106	6. Design Patterns with First-Class Functions	173
109	Case Study: Refactoring Strategy	174
109	Classic Strategy	174
110	Function-Oriented Strategy	178
112	Choosing the Best Strategy: Simple Approach	181
113	Finding Strategies in a Module	182
114	Command	184
115	Chapter Summary	185
118	Further Reading	186
121	7. Function Decorators and Closures	189
122	Decorators 101	190
126	When Python Executes Decorators	191
129	Decorator-Enhanced Strategy Pattern	193
131	Variable Scope Rules	195
132	Closures	198
134	The <code>nonlocal</code> Declaration	201
134	Implementing a Simple Decorator	203
135	How It Works	204
138	Decorators in the Standard Library	206
139	Memoization with <code>functools.lru_cache</code>	206
	Generic Functions with Single Dispatch	208
	Stacked Decorators	212
	Parameterized Decorators	212
	A Parameterized Registration Decorator	213
142	The Parameterized Clock Decorator	215
146	Chapter Summary	218
147	Further Reading	218

Part IV. Object-Oriented Idioms

8. Object References, Mutability, and Recycling.....	225
Variables Are Not Boxes	226
Identity, Equality, and Aliases	227
Choosing Between == and is	229
The Relative Immutability of Tuples	230
Copies Are Shallow by Default	231
Deep and Shallow Copies of Arbitrary Objects	234
Function Parameters as References	235
Mutable Types as Parameter Defaults: Bad Idea	236
Defensive Programming with Mutable Parameters	239
del and Garbage Collection	241
Weak References	242
The WeakValueDictionary Skit	244
Limitations of Weak References	246
Tricks Python Plays with Immutables	247
Chapter Summary	249
Further Reading	250
9. A Pythonic Object.....	255
Object Representations	256
Vector Class Redux	256
An Alternative Constructor	260
classmethod Versus staticmethod	260
Formatted Displays	262
A Hashable Vector2d	266
Private and “Protected” Attributes in Python	272
Saving Space with the <code>__slots__</code> Class Attribute	274
The Problems with <code>__slots__</code>	276
Overriding Class Attributes	277
Chapter Summary	279
Further Reading	281
10. Sequence Hacking, Hashing, and Slicing.....	285
Vector: A User-Defined Sequence Type	286
Vector Take #1: Vector2d Compatible	286
Protocols and Duck Typing	289
Vector Take #2: A Sliceable Sequence	290
How Slicing Works	291
A Slice-Aware <code>__getitem__</code>	293
Vector Take #3: Dynamic Attribute Access	295

Vector Take #4: Hashing and a Faster ==	299
Vector Take #5: Formatting	305
Chapter Summary	312
Further Reading	313
11. Interfaces: From Protocols to ABCs.....	319
Interfaces and Protocols in Python Culture	320
Python Digs Sequences	322
Monkey-Patching to Implement a Protocol at Runtime	324
Alex Martelli's Waterfowl	326
Subclassing an ABC	331
ABCs in the Standard Library	333
ABCs in collections.abc	333
The Numbers Tower of ABCs	335
Defining and Using an ABC	336
ABC Syntax Details	340
Subclassing the Tombola ABC	341
A Virtual Subclass of Tombola	344
How the Tombola Subclasses Were Tested	347
Usage of register in Practice	350
Geese Can Behave as Ducks	351
Chapter Summary	352
Further Reading	355
12. Inheritance: For Good or For Worse.....	361
Subclassing Built-In Types Is Tricky	362
Multiple Inheritance and Method Resolution Order	365
Multiple Inheritance in the Real World	371
Coping with Multiple Inheritance	373
1. Distinguish Interface Inheritance from Implementation Inheritance	374
2. Make Interfaces Explicit with ABCs	374
3. Use Mixins for Code Reuse	374
4. Make Mixins Explicit by Naming	374
5. An ABC May Also Be a Mixin; The Reverse Is Not True	374
6. Don't Subclass from More Than One Concrete Class	375
7. Provide Aggregate Classes to Users	375
8. "Favor Object Composition Over Class Inheritance."	376
Tkinter: The Good, the Bad, and the Ugly	376
A Modern Example: Mixins in Django Generic Views	377
Chapter Summary	381
Further Reading	382

13. Operator Overloading: Doing It Right.....	385
Operator Overloading 101	386
Unary Operators	386
Overloading + for Vector Addition	389
Overloading * for Scalar Multiplication	394
Rich Comparison Operators	399
Augmented Assignment Operators	403
Chapter Summary	408
Further Reading	409

Part V. Control Flow

14. Iterables, Iterators, and Generators.....	415
Sentence Take #1: A Sequence of Words	416
Why Sequences Are Iterable: The iter Function	418
Iterables Versus Iterators	420
Sentence Take #2: A Classic Iterator	424
Making Sentence an Iterator: Bad Idea	425
Sentence Take #3: A Generator Function	426
How a Generator Function Works	428
Sentence Take #4: A Lazy Implementation	431
Sentence Take #5: A Generator Expression	432
Generator Expressions: When to Use Them	434
Another Example: Arithmetic Progression Generator	435
Arithmetic Progression with itertools	438
Generator Functions in the Standard Library	439
New Syntax in Python 3.3: yield from	449
Iterable Reducing Functions	450
A Closer Look at the iter Function	452
Case Study: Generators in a Database Conversion Utility	453
Generators as Coroutines	455
Chapter Summary	455
Further Reading	456
15. Context Managers and else Blocks.....	463
Do This, Then That: else Blocks Beyond if	464
Context Managers and with Blocks	466
The contextlib Utilities	470
Using @contextmanager	471
Chapter Summary	476
Further Reading	476

16. Coroutines.....	479
How Coroutines Evolved from Generators	480
Basic Behavior of a Generator Used as a Coroutine	481
Example: Coroutine to Compute a Running Average	484
Decorators for Coroutine Priming	486
Coroutine Termination and Exception Handling	488
Returning a Value from a Coroutine	491
Using yield from	494
The Meaning of yield from	500
Use Case: Coroutines for Discrete Event Simulation	506
About Discrete Event Simulations	507
The Taxi Fleet Simulation	508
Chapter Summary	516
Further Reading	518
17. Concurrency with Futures.....	523
Example: Web Downloads in Three Styles	523
A Sequential Download Script	525
Downloading with concurrent.futures	528
Where Are the Futures?	529
Blocking I/O and the GIL	533
Launching Processes with concurrent.futures	534
Experimenting with Executor.map	536
Downloads with Progress Display and Error Handling	539
Error Handling in the flags2 Examples	544
Using futures.as_completed	547
Threading and Multiprocessing Alternatives	550
Chapter Summary	550
Further Reading	551
18. Concurrency with asyncio.....	557
Thread Versus Coroutine: A Comparison	559
asyncio.Future: Nonblocking by Design	566
Yielding from Futures, Tasks, and Coroutines	567
Downloading with asyncio and aiohttp	568
Running Circles Around Blocking Calls	573
Enhancing the asyncio downloader Script	575
Using asyncio.as_completed	576
Using an Executor to Avoid Blocking the Event Loop	582
From Callbacks to Futures and Coroutines	583
Doing Multiple Requests for Each Download	586
Writing asyncio Servers	589

An asyncio TCP Server	590
An aiohttp Web Server	595
Smarter Clients for Better Concurrency	598
Chapter Summary	599
Further Reading	601

Part VI. Metaprogramming

19. Dynamic Attributes and Properties	607
Data Wrangling with Dynamic Attributes	608
Exploring JSON-Like Data with Dynamic Attributes	610
The Invalid Attribute Name Problem	614
Flexible Object Creation with <code>__new__</code>	615
Restructuring the OSCON Feed with <code>shelve</code>	617
Linked Record Retrieval with Properties	621
Using a Property for Attribute Validation	628
LineItem Take #1: Class for an Item in an Order	628
LineItem Take #2: A Validating Property	629
A Proper Look at Properties	630
Properties Override Instance Attributes	632
Property Documentation	634
Coding a Property Factory	636
Handling Attribute Deletion	639
Essential Attributes and Functions for Attribute Handling	640
Special Attributes that Affect Attribute Handling	640
Built-In Functions for Attribute Handling	641
Special Methods for Attribute Handling	642
Chapter Summary	643
Further Reading	644
20. Attribute Descriptors	649
Descriptor Example: Attribute Validation	650
LineItem Take #3: A Simple Descriptor	650
LineItem Take #4: Automatic Storage Attribute Names	655
LineItem Take #5: A New Descriptor Type	661
Overriding Versus Nonoverriding Descriptors	665
Overriding Descriptor	667
Overriding Descriptor Without <code>__get__</code>	668
Nonoverriding Descriptor	669
Overwriting a Descriptor in the Class	670
Methods Are Descriptors	671

Descriptor Usage Tips	673
Descriptor docstring and Overriding Deletion	675
Chapter Summary	676
Further Reading	677
21. Class Metaprogramming	681
A Class Factory	682
A Class Decorator for Customizing Descriptors	685
What Happens When: Import Time Versus Runtime	688
The Evaluation Time Exercises	689
Metaclasses 101	693
The Metaclass Evaluation Time Exercise	695
A Metaclass for Customizing Descriptors	699
The Metaclass <code>__prepare__</code> Special Method	701
Classes as Objects	703
Chapter Summary	704
Further Reading	706
Afterword	709
A. Support Scripts	713
Python Jargon	741
Index	751