

SECOND EDITION

---

# Fluent Python

*Clear, Concise, and  
Effective Programming*

*Luciano Ramalho*

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

# Fluent Python

by Luciano Ramalho

Copyright © 2022 Luciano Ramalho. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Acquisitions Editor:** Amanda Quinn

**Development Editor:** Jeff Bleiel

**Production Editor:** Daniel Elfanbaum

**Copyeditor:** Sonia Saruba

**Proofreader:** Kim Cofer

**Indexer:** Judith McConville

**Interior Designer:** David Futato

**Cover Designer:** Karen Montgomery

**Illustrator:** Kate Dullea

April 2022:                      Second Edition

## Revision History for the Second Edition

2022-03-31:    First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781492056355> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Fluent Python*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-492-05635-5

[LSI]

---

# Table of Contents

Preface.....	xix
--------------	-----

---

## Part I. Data Structures

<b>1. The Python Data Model.....</b>	<b>3</b>
What's New in This Chapter	4
A Pythonic Card Deck	5
How Special Methods Are Used	8
Emulating Numeric Types	9
String Representation	12
Boolean Value of a Custom Type	13
Collection API	14
Overview of Special Methods	15
Why len Is Not a Method	17
Chapter Summary	18
Further Reading	18
<b>2. An Array of Sequences.....</b>	<b>21</b>
What's New in This Chapter	22
Overview of Built-In Sequences	22
List Comprehensions and Generator Expressions	25
List Comprehensions and Readability	25
Listcomps Versus map and filter	27
Cartesian Products	27
Generator Expressions	29
Tuples Are Not Just Immutable Lists	30
Tuples as Records	30

Tuples as Immutable Lists	32
Comparing Tuple and List Methods	34
Unpacking Sequences and Iterables	35
Using * to Grab Excess Items	36
Unpacking with * in Function Calls and Sequence Literals	37
Nested Unpacking	37
Pattern Matching with Sequences	38
Pattern Matching Sequences in an Interpreter	43
Slicing	47
Why Slices and Ranges Exclude the Last Item	47
Slice Objects	48
Multidimensional Slicing and Ellipsis	49
Assigning to Slices	50
Using + and * with Sequences	50
Building Lists of Lists	51
Augmented Assignment with Sequences	53
A += Assignment Puzzler	54
list.sort Versus the sorted Built-In	56
When a List Is Not the Answer	59
Arrays	59
Memory Views	62
NumPy	64
Deque and Other Queues	67
Chapter Summary	70
Further Reading	71
<b>3. Dictionaries and Sets</b>	<b>77</b>
What's New in This Chapter	78
Modern dict Syntax	78
dict Comprehensions	79
Unpacking Mappings	80
Merging Mappings with	80
Pattern Matching with Mappings	81
Standard API of Mapping Types	83
What Is Hashable	84
Overview of Common Mapping Methods	85
Inserting or Updating Mutable Values	87
Automatic Handling of Missing Keys	90
defaultdict: Another Take on Missing Keys	90
The __missing__ Method	91
Inconsistent Usage of __missing__ in the Standard Library	94
Variations of dict	95

collections.OrderedDict	95
collections.ChainMap	95
collections.Counter	96
shelve.Shelf	97
Subclassing UserDict Instead of dict	97
Immutable Mappings	99
Dictionary Views	101
Practical Consequences of How dict Works	102
Set Theory	103
Set Literals	105
Set Comprehensions	106
Practical Consequences of How Sets Work	107
Set Operations	107
Set Operations on dict Views	110
Chapter Summary	112
Further Reading	113
<b>4. Unicode Text Versus Bytes.....</b>	<b>117</b>
What's New in This Chapter	118
Character Issues	118
Byte Essentials	120
Basic Encoders/Decoders	123
Understanding Encode/Decode Problems	125
Coping with UnicodeEncodeError	125
Coping with UnicodeDecodeError	126
SyntaxError When Loading Modules with Unexpected Encoding	128
How to Discover the Encoding of a Byte Sequence	128
BOM: A Useful Gremlin	129
Handling Text Files	131
Beware of Encoding Defaults	134
Normalizing Unicode for Reliable Comparisons	140
Case Folding	142
Utility Functions for Normalized Text Matching	143
Extreme "Normalization": Taking Out Diacritics	144
Sorting Unicode Text	148
Sorting with the Unicode Collation Algorithm	150
The Unicode Database	150
Finding Characters by Name	151
Numeric Meaning of Characters	153
Dual-Mode str and bytes APIs	155
str Versus bytes in Regular Expressions	155
str Versus bytes in os Functions	156

Chapter Summary	157
Further Reading	158
<b>5. Data Class Builders.....</b>	<b>163</b>
What's New in This Chapter	164
Overview of Data Class Builders	164
Main Features	167
Classic Named Tuples	169
Typed Named Tuples	172
Type Hints 101	173
No Runtime Effect	173
Variable Annotation Syntax	174
The Meaning of Variable Annotations	175
More About @dataclass	179
Field Options	180
Post-init Processing	183
Typed Class Attributes	185
Initialization Variables That Are Not Fields	186
@dataclass Example: Dublin Core Resource Record	187
Data Class as a Code Smell	190
Data Class as Scaffolding	191
Data Class as Intermediate Representation	191
Pattern Matching Class Instances	192
Simple Class Patterns	192
Keyword Class Patterns	193
Positional Class Patterns	194
Chapter Summary	195
Further Reading	196
<b>6. Object References, Mutability, and Recycling.....</b>	<b>201</b>
What's New in This Chapter	202
Variables Are Not Boxes	202
Identity, Equality, and Aliases	204
Choosing Between == and is	206
The Relative Immutability of Tuples	207
Copies Are Shallow by Default	208
Deep and Shallow Copies of Arbitrary Objects	211
Function Parameters as References	213
Mutable Types as Parameter Defaults: Bad Idea	214
Defensive Programming with Mutable Parameters	216
del and Garbage Collection	219
Tricks Python Plays with Immutables	221

Chapter Summary	223
Further Reading	224

---

## Part II. Functions as Objects

<b>7. Functions as First-Class Objects.....</b>	<b>231</b>
What's New in This Chapter	232
Treating a Function Like an Object	232
Higher-Order Functions	234
Modern Replacements for map, filter, and reduce	235
Anonymous Functions	236
The Nine Flavors of Callable Objects	237
User-Defined Callable Types	239
From Positional to Keyword-Only Parameters	240
Positional-Only Parameters	242
Packages for Functional Programming	243
The operator Module	243
Freezing Arguments with functools.partial	247
Chapter Summary	249
Further Reading	250
<b>8. Type Hints in Functions.....</b>	<b>253</b>
What's New in This Chapter	254
About Gradual Typing	254
Gradual Typing in Practice	255
Starting with Mypy	256
Making Mypy More Strict	257
A Default Parameter Value	258
Using None as a Default	260
Types Are Defined by Supported Operations	260
Types Usable in Annotations	266
The Any Type	266
Simple Types and Classes	269
Optional and Union Types	270
Generic Collections	271
Tuple Types	274
Generic Mappings	276
Abstract Base Classes	278
Iterable	280
Parameterized Generics and TypeVar	282
Static Protocols	286

Callable	291
NoReturn	294
Annotating Positional Only and Variadic Parameters	295
Imperfect Typing and Strong Testing	296
Chapter Summary	297
Further Reading	298
<b>9. Decorators and Closures. ....</b>	<b>303</b>
What's New in This Chapter	304
Decorators 101	304
When Python Executes Decorators	306
Registration Decorators	308
Variable Scope Rules	308
Closures	311
The nonlocal Declaration	315
Variable Lookup Logic	316
Implementing a Simple Decorator	317
How It Works	318
Decorators in the Standard Library	320
Memoization with functools.cache	320
Using lru_cache	323
Single Dispatch Generic Functions	324
Parameterized Decorators	329
A Parameterized Registration Decorator	329
The Parameterized Clock Decorator	332
A Class-Based Clock Decorator	335
Chapter Summary	336
Further Reading	336
<b>10. Design Patterns with First-Class Functions. ....</b>	<b>341</b>
What's New in This Chapter	342
Case Study: Refactoring Strategy	342
Classic Strategy	342
Function-Oriented Strategy	347
Choosing the Best Strategy: Simple Approach	350
Finding Strategies in a Module	351
Decorator-Enhanced Strategy Pattern	353
The Command Pattern	355
Chapter Summary	357
Further Reading	358



---

## Part III. Classes and Protocols

<b>11. A Pythonic Object.....</b>	<b>363</b>
What's New in This Chapter	364
Object Representations	364
Vector Class Redux	365
An Alternative Constructor	368
classmethod Versus staticmethod	369
Formatted Displays	370
A Hashable Vector2d	374
Supporting Positional Pattern Matching	377
Complete Listing of Vector2d, Version 3	378
Private and "Protected" Attributes in Python	382
Saving Memory with __slots__	384
Simple Measure of __slot__ Savings	387
Summarizing the Issues with __slots__	388
Overriding Class Attributes	389
Chapter Summary	391
Further Reading	392
<b>12. Special Methods for Sequences.....</b>	<b>397</b>
What's New in This Chapter	398
Vector: A User-Defined Sequence Type	398
Vector Take #1: Vector2d Compatible	399
Protocols and Duck Typing	402
Vector Take #2: A Sliceable Sequence	403
How Slicing Works	404
A Slice-Aware __getitem__	406
Vector Take #3: Dynamic Attribute Access	407
Vector Take #4: Hashing and a Faster ==	411
Vector Take #5: Formatting	418
Chapter Summary	425
Further Reading	426
<b>13. Interfaces, Protocols, and ABCs.....</b>	<b>431</b>
The Typing Map	432
What's New in This Chapter	433
Two Kinds of Protocols	434
Programming Ducks	435
Python Digs Sequences	436
Monkey Patching: Implementing a Protocol at Runtime	438
Defensive Programming and "Fail Fast"	440

Goose Typing	442
Subclassing an ABC	447
ABCs in the Standard Library	449
Defining and Using an ABC	451
ABC Syntax Details	457
Subclassing an ABC	458
A Virtual Subclass of an ABC	460
Usage of register in Practice	463
Structural Typing with ABCs	464
Static Protocols	466
The Typed double Function	466
Runtime Checkable Static Protocols	468
Limitations of Runtime Protocol Checks	471
Supporting a Static Protocol	472
Designing a Static Protocol	474
Best Practices for Protocol Design	476
Extending a Protocol	477
The numbers ABCs and Numeric Protocols	478
Chapter Summary	481
Further Reading	482

<b>14. Inheritance: For Better or for Worse. ....</b>	<b>487</b>
What's New in This Chapter	488
The super() Function	488
Subclassing Built-In Types Is Tricky	490
Multiple Inheritance and Method Resolution Order	494
Mixin Classes	500
Case-Insensitive Mappings	500
Multiple Inheritance in the Real World	502
ABCs Are Mixins Too	502
ThreadingMixin and ForkingMixin	503
Django Generic Views Mixins	504
Multiple Inheritance in Tkinter	507
Coping with Inheritance	510
Favor Object Composition over Class Inheritance	510
Understand Why Inheritance Is Used in Each Case	510
Make Interfaces Explicit with ABCs	511
Use Explicit Mixins for Code Reuse	511
Provide Aggregate Classes to Users	511
Subclass Only Classes Designed for Subclassing	512
Avoid Subclassing from Concrete Classes	513
Tkinter: The Good, the Bad, and the Ugly	513

Chapter Summary	514
Further Reading	515
<b>15. More About Type Hints. ....</b>	<b>519</b>
What's New in This Chapter	519
Overloaded Signatures	520
Max Overload	521
Takeaways from Overloading max	525
TypedDict	526
Type Casting	534
Reading Type Hints at Runtime	537
Problems with Annotations at Runtime	538
Dealing with the Problem	540
Implementing a Generic Class	541
Basic Jargon for Generic Types	544
Variance	544
An Invariant Dispenser	545
A Covariant Dispenser	546
A Contravariant Trash Can	547
Variance Review	549
Implementing a Generic Static Protocol	552
Chapter Summary	554
Further Reading	555
<b>16. Operator Overloading. ....</b>	<b>561</b>
What's New in This Chapter	562
Operator Overloading 101	562
Unary Operators	563
Overloading + for Vector Addition	566
Overloading * for Scalar Multiplication	572
Using @ as an Infix Operator	574
Wrapping-Up Arithmetic Operators	576
Rich Comparison Operators	577
Augmented Assignment Operators	580
Chapter Summary	585
Further Reading	587

---

## Part IV. Control Flow

<b>17. Iterators, Generators, and Classic Coroutines. ....</b>	<b>593</b>
What's New in This Chapter	594

A Sequence of Words	594
Why Sequences Are Iterable: The iter Function	596
Using iter with a Callable	598
Iterables Versus Iterators	599
Sentence Classes with __iter__	603
Sentence Take #2: A Classic Iterator	603
Don't Make the Iterable an Iterator for Itself	605
Sentence Take #3: A Generator Function	606
How a Generator Works	607
Lazy Sentences	610
Sentence Take #4: Lazy Generator	610
Sentence Take #5: Lazy Generator Expression	611
When to Use Generator Expressions	613
An Arithmetic Progression Generator	615
Arithmetic Progression with itertools	618
Generator Functions in the Standard Library	619
Iterable Reducing Functions	630
Subgenerators with yield from	632
Reinventing chain	633
Traversing a Tree	634
Generic Iterable Types	639
Classic Coroutines	641
Example: Coroutine to Compute a Running Average	643
Returning a Value from a Coroutine	646
Generic Type Hints for Classic Coroutines	650
Chapter Summary	652
Further Reading	652
<b>18. with, match, and else Blocks.....</b>	<b>657</b>
What's New in This Chapter	658
Context Managers and with Blocks	658
The contextlib Utilities	663
Using @contextmanager	664
Pattern Matching in lis.py: A Case Study	669
Scheme Syntax	669
Imports and Types	671
The Parser	671
The Environment	673
The REPL	675
The Evaluator	676
Procedure: A Class Implementing a Closure	685
Using OR-patterns	686

Do This, Then That: else Blocks Beyond if	687
Chapter Summary	689
Further Reading	690
<b>19. Concurrency Models in Python.....</b>	<b>695</b>
What's New in This Chapter	696
The Big Picture	696
A Bit of Jargon	697
Processes, Threads, and Python's Infamous GIL	699
A Concurrent Hello World	701
Spinner with Threads	701
Spinner with Processes	704
Spinner with Coroutines	706
Supervisors Side-by-Side	711
The Real Impact of the GIL	713
Quick Quiz	713
A Homegrown Process Pool	716
Process-Based Solution	718
Understanding the Elapsed Times	718
Code for the Multicore Prime Checker	719
Experimenting with More or Fewer Processes	723
Thread-Based Nonsolution	724
Python in the Multicore World	725
System Administration	726
Data Science	727
Server-Side Web/Mobile Development	728
WSGI Application Servers	730
Distributed Task Queues	732
Chapter Summary	733
Further Reading	734
Concurrency with Threads and Processes	734
The GIL	736
Concurrency Beyond the Standard Library	736
Concurrency and Scalability Beyond Python	738
<b>20. Concurrent Executors.....</b>	<b>743</b>
What's New in This Chapter	743
Concurrent Web Downloads	744
A Sequential Download Script	746
Downloading with concurrent.futures	749
Where Are the Futures?	751
Launching Processes with concurrent.futures	754

Multicore Prime Checker Redux	755
Experimenting with Executor.map	758
Downloads with Progress Display and Error Handling	762
Error Handling in the flags2 Examples	766
Using futures.as_completed	769
Chapter Summary	772
Further Reading	772
<b>21. Asynchronous Programming.....</b>	<b>775</b>
What's New in This Chapter	776
A Few Definitions	777
An async Example: Probing Domains	778
Guido's Trick to Read Asynchronous Code	780
New Concept: Awaitable	781
Downloading with asyncio and HTTPX	782
The Secret of Native Coroutines: Humble Generators	784
The All-or-Nothing Problem	785
Asynchronous Context Managers	786
Enhancing the asyncio Downloader	787
Using asyncio.as_completed and a Thread	788
Throttling Requests with a Semaphore	790
Making Multiple Requests for Each Download	794
Delegating Tasks to Executors	797
Writing asyncio Servers	799
A FastAPI Web Service	800
An asyncio TCP Server	804
Asynchronous Iteration and Asynchronous Iterables	811
Asynchronous Generator Functions	812
Async Comprehensions and Async Generator Expressions	818
async Beyond asyncio: Curio	821
Type Hinting Asynchronous Objects	824
How Async Works and How It Doesn't	825
Running Circles Around Blocking Calls	825
The Myth of I/O-Bound Systems	826
Avoiding CPU-Bound Traps	826
Chapter Summary	827
Further Reading	828

---

## Part V. Metaprogramming

<b>22. Dynamic Attributes and Properties.....</b>	<b>835</b>
What's New in This Chapter	836
Data Wrangling with Dynamic Attributes	836
Exploring JSON-Like Data with Dynamic Attributes	838
The Invalid Attribute Name Problem	842
Flexible Object Creation with <code>__new__</code>	843
Computed Properties	845
Step 1: Data-Driven Attribute Creation	846
Step 2: Property to Retrieve a Linked Record	848
Step 3: Property Overriding an Existing Attribute	852
Step 4: Bespoke Property Cache	853
Step 5: Caching Properties with <code>functools</code>	855
Using a Property for Attribute Validation	857
LineItem Take #1: Class for an Item in an Order	857
LineItem Take #2: A Validating Property	858
A Proper Look at Properties	860
Properties Override Instance Attributes	861
Property Documentation	864
Coding a Property Factory	865
Handling Attribute Deletion	868
Essential Attributes and Functions for Attribute Handling	869
Special Attributes that Affect Attribute Handling	870
Built-In Functions for Attribute Handling	870
Special Methods for Attribute Handling	871
Chapter Summary	873
Further Reading	873
<b>23. Attribute Descriptors.....</b>	<b>879</b>
What's New in This Chapter	880
Descriptor Example: Attribute Validation	880
LineItem Take #3: A Simple Descriptor	880
LineItem Take #4: Automatic Naming of Storage Attributes	887
LineItem Take #5: A New Descriptor Type	889
Overriding Versus Nonoverriding Descriptors	892
Overriding Descriptors	894
Overriding Descriptor Without <code>__get__</code>	895
Nonoverriding Descriptor	896
Overwriting a Descriptor in the Class	897
Methods Are Descriptors	898
Descriptor Usage Tips	900

Descriptor Docstring and Overriding Deletion	902
Chapter Summary	903
Further Reading	904
<b>24. Class Metaprogramming.....</b>	<b>907</b>
What's New in This Chapter	908
Classes as Objects	908
type: The Built-In Class Factory	909
A Class Factory Function	911
Introducing <code>__init_subclass__</code>	914
Why <code>__init_subclass__</code> Cannot Configure <code>__slots__</code>	921
Enhancing Classes with a Class Decorator	922
What Happens When: Import Time Versus Runtime	925
Evaluation Time Experiments	926
Metaclasses 101	931
How a Metaclass Customizes a Class	933
A Nice Metaclass Example	934
Metaclass Evaluation Time Experiment	937
A Metaclass Solution for Checked	942
Metaclasses in the Real World	947
Modern Features Simplify or Replace Metaclasses	947
Metaclasses Are Stable Language Features	948
A Class Can Only Have One Metaclass	948
Metaclasses Should Be Implementation Details	949
A Metaclass Hack with <code>__prepare__</code>	950
Wrapping Up	952
Chapter Summary	953
Further Reading	954
<b>Afterword.....</b>	<b>959</b>
<b>Index.....</b>	<b>963</b>