

PYTHON TRICKS THE BOOK

A Buffet
of Awesome
Python
Features



Dan Bader

Python Tricks: The Book

Dan Bader

For online information and ordering of this and other books by Dan Bader, please visit realpython.com. For more information, please contact Dan Bader at info@realpython.com.

Copyright © Dan Bader (realpython.com), 2016–2018

ISBN: 9781775093305 (paperback)

ISBN: 9781775093312 (electronic)

Cover design by Anja Pircher Design (anjapircher.com)

“Python” and the Python logos are trademarks or registered trademarks of the Python Software Foundation, used by Dan Bader with permission from the Foundation.

Thank you for downloading this ebook. This ebook is licensed for your personal enjoyment only. This ebook may not be re-sold or given away to other people. If you would like to share this book with another person, please purchase an additional copy for each recipient. If you’re reading this book and did not purchase it, or it was not purchased for your use only, then please return to realpython.com/pytricks-book and purchase your own copy. Thank you for respecting the hard work behind this book.

What Pythonistas Say About *Python Tricks: The Book*

"I love love love the book. It's like having a seasoned tutor explaining, well, tricks! I'm learning Python on the job and I'm coming from Powershell, which I learned on the job—so lots of new, great stuff. Whenever I get stuck in Python (usually with Flask blueprints or I feel like my code could be more Pythonic) I post questions in our internal Python chat room.

I'm often amazed at some of the answers co-workers give me. Dict comprehensions, lambdas, and generators often pepper their feedback. I am always impressed and yet flabbergasted at how powerful Python is when you know these tricks and can implement them correctly.

Your book was exactly what I wanted to help get me from a bewildered Powershell scripter to someone who knows how and when to use these Pythonic 'tricks' everyone has been talking about.

As someone who doesn't have my degree in CS it's nice to have the text to explain things that others might have learned when they were classically educated. I am really enjoying the book and am subscribed to the emails as well, which is how I found out about the book."

— **Daniel Meyer**, Sr. Desktop Administrator at Tesla Inc.

"I first heard about your book from a co-worker who wanted to trick me with your example of how dictionaries are built. I was almost 100% sure about the reason why the end product was a much smaller/simpler dictionary but I must confess that I did not expect the outcome :)

He showed me the book via video conferencing and I sort of skimmed through it as he flipped the pages for me, and I was immediately curious to read more.

That same afternoon I purchased my own copy and proceeded to read your explanation for the way dictionaries are created in Python and later that day, as I met a different co-worker for coffee, I used the same trick on him :)

He then sprung a different question on the same principle, and because of the way you explained things in your book, I was able to not simply guess the result but to correctly answer what the outcome would be. That means that you did a great job at explaining things :)

I am not new in Python and some of the concepts in some of the chapters are not new to me, but I must say that I do get something out of every chapter so far, so kudos for writing a very nice book and for doing a fantastic job at explaining concepts behind the tricks! I'm very much looking forward to the updates and I will certainly let my friends and co-workers know about your book."

— **Og Maciel**, Python Developer at Red Hat

"I really enjoyed reading Dan's book. He explains important Python aspects with clear examples (using two twin cats to explain 'is' vs '==' for example).

It is not just code samples, it discusses relevant implementation details comprehensibly. What really matters though is that this book makes you write better Python code!

The book is actually responsible for recent new good Python habits I picked up, for example: using custom exceptions and ABC's (I found Dan's blog searching for abstract classes.) These new learnings alone are worth the price."

— **Bob Belderbos**, Engineer at Oracle & Co-Founder of PyBites

This is a sample from “Python Tricks: The Book”

The full version of the book includes many more Python Tricks that will teach you the depths and quirks of Python with fun and easy to understand examples and explanations.

If you enjoyed the sample chapters you can purchase a full version of the book at realpython.com/pytricks-book.

Contents

Contents	6
Foreword	9
1 Introduction	11
1.1 What’s a Python Trick?	11
1.2 What This Book Will Do for You	13
1.3 How to Read This Book	14
2 Patterns for Cleaner Python	15
2.1 Covering Your A** With Assertions	16
2.2 Complacent Comma Placement	25
2.3 Context Managers and the with Statement	29
2.4 Underscores, Dunders, and More	36
2.5 A Shocking Truth About String Formatting	48
2.6 “The Zen of Python” Easter Egg	56
3 Effective Functions	57
3.1 Python’s Functions Are First-Class	58
3.2 Lambdas Are Single-Expression Functions	68
3.3 The Power of Decorators	73
3.4 Fun With *args and **kwargs	86
3.5 Function Argument Unpacking	91
3.6 Nothing to Return Here	94

4	Classes & OOP	97
4.1	Object Comparisons: “is” vs “==”	98
4.2	String Conversion (Every Class Needs a <code>__repr__</code>)	101
4.3	Defining Your Own Exception Classes	111
4.4	Cloning Objects for Fun and Profit	116
4.5	Abstract Base Classes Keep Inheritance in Check	124
4.6	What Namedtuples Are Good For	128
4.7	Class vs Instance Variable Pitfalls	136
4.8	Instance, Class, and Static Methods Demystified	143
5	Common Data Structures in Python	153
5.1	Dictionaries, Maps, and Hashtables	156
5.2	Array Data Structures	163
5.3	Records, Structs, and Data Transfer Objects	173
5.4	Sets and Multisets	185
5.5	Stacks (LIFOs)	189
5.6	Queues (FIFOs)	195
5.7	Priority Queues	201
6	Looping & Iteration	205
6.1	Writing Pythonic Loops	206
6.2	Comprehending Comprehensions	210
6.3	List Slicing Tricks and the Sushi Operator	214
6.4	Beautiful Iterators	218
6.5	Generators Are Simplified Iterators	231
6.6	Generator Expressions	239
6.7	Iterator Chains	246
7	Dictionary Tricks	250
7.1	Dictionary Default Values	251
7.2	Sorting Dictionaries for Fun and Profit	255
7.3	Emulating Switch/Case Statements With Dicts	259
7.4	The Craziest Dict Expression in the West	264
7.5	So Many Ways to Merge Dictionaries	271
7.6	Dictionary Pretty-Printing	274

8	Pythonic Productivity Techniques	277
8.1	Exploring Python Modules and Objects	278
8.2	Isolating Project Dependencies With Virtualenv . . .	282
8.3	Peeking Behind the Bytecode Curtain	288
9	Closing Thoughts	293
9.1	Free Weekly Tips for Python Developers	295
9.2	PythonistaCafe: A Community for Python Developers	296

Foreword

It's been almost ten years since I first got acquainted with Python as a programming language. When I first learned Python many years ago, it was with a little reluctance. I had been programming in a different language before, and all of the sudden at work, I was assigned to a different team where everyone used Python. That was the beginning of my own Python journey.

When I was first introduced to Python, I was told that it was going to be easy, that I should be able to pick it up quickly. When I asked my colleagues for resources for learning Python, all they gave me was a link to Python's official documentation. Reading the documentation was confusing at first, and it really took me a while before I even felt comfortable navigating through it. Often I found myself needing to look for answers in StackOverflow.

Coming from a different programming language, I wasn't looking for just any resource for learning how to program or what classes and objects are. I was looking for specific resources that would teach me the features of Python, what sets it apart, and how writing in Python is different than writing code in another language.

It really has taken me many years to fully appreciate this language. As I read Dan's book, I kept thinking that I wished I had access to a book like this when I started learning Python many years ago.

For example, one of the many unique Python features that surprised me at first were list comprehensions. As Dan mentions in the book,

a tell of someone who just came to Python from a different language is the way they use for-loops. I recall one of the earliest code review comments I got when I started programming in Python was, “Why not use list comprehension here?” Dan explains this concept clearly in section 6, starting by showing how to loop the Pythonic way and building it all the way up to iterators and generators.

In chapter 2.5, Dan discusses the different ways to do string formatting in Python. String formatting is one of those things that defy the Zen of Python, that there should only be one obvious way to do things. Dan shows us the different ways, including my favorite new addition to the language, the f-strings, and he also explains the pros and cons of each method.

The Pythonic Productivity Techniques section is another great resource. It covers aspects beyond the Python programming language, and also includes tips on how to debug your programs, how to manage the dependencies, and gives you a peek inside Python bytecode.

It truly is an honor and my pleasure to introduce this book, Python Tricks, by my friend, Dan Bader.

By contributing to Python as a CPython core developer, I get connected to many members of the community. In my journey, I found mentors, allies, and made many new friends. They remind me that Python is not just about the code, Python is a community.

Mastering Python programming isn’t just about grasping the theoretical aspects of the language. It’s just as much about understanding and adopting the conventions and best practices used by its community.

Dan’s book will help you on this journey. I’m convinced that you’ll be more confident when writing Python programs after reading it.

— **Mariatta Wijaya**, Python Core Developer (mariatta.ca)

Chapter 1

Introduction

1.1 What's a Python Trick?

Python Trick: *A short Python code snippet meant as a teaching tool. A Python Trick either teaches an aspect of Python with a simple illustration, or it serves as a motivating example, enabling you to dig deeper and develop an intuitive understanding.*

Python Tricks started out as a short series of code screenshots that I shared on Twitter for a week. To my surprise, they got rave responses and were shared and retweeted for days on end.

More and more developers started asking me for a way to “get the whole series.” Actually, I only had a few of these tricks lined up, spanning a variety of Python-related topics. There wasn’t a master plan behind them. They were just a fun little Twitter experiment.

But from these inquiries I got the sense that my short-and-sweet code examples would be worth exploring as a teaching tool. Eventually I set out to create a few more Python Tricks and shared them in an email series. Within a few days, several hundred Python developers had signed up and I was just blown away by that response.

Over the following days and weeks, a steady stream of Python developers reached out to me. They thanked me for making a part of the language they were struggling to understand *click* for them. Hearing this feedback felt awesome. I thought these Python Tricks were just code screenshots, but so many developers were getting a lot of value out of them.

That's when I decided to double down on my Python Tricks experiment and expanded it into a series of around 30 emails. Each of these was still just a a headline and a code screenshot, and I soon realized the limits of that format. Around this time, a blind Python developer emailed me, disappointed to find that these Python Tricks were delivered as images he couldn't read with his screen reader.

Clearly, I needed to invest more time into this project to make it more appealing and more accessible to a wider audience. So, I sat down to re-create the whole series of Python Tricks emails in plain text and with proper HTML-based syntax highlighting. That new iteration of Python Tricks chugged along nicely for a while. Based on the responses I got, developers seemed happy they could finally copy and paste the code samples in order to play around with them.

As more and more developers signed up for the email series, I started noticing a pattern in the replies and questions I received. Some Tricks worked well as motivational examples by themselves. However, for the more complex ones there was no narrator to guide readers or to give them additional resources to develop a deeper understanding.

Let's just say this was another big area of improvement. My mission statement for realpython.com is to *help Python developers become more awesome*—and this was clearly an opportunity to get closer to that goal.

I decided to take the best and most valuable Python Tricks from the email course, and I started writing a new kind of Python book around them:

- A book that teaches the coolest aspects of the language with short and easy-to-digest examples.
- A book that works like a buffet of awesome Python features (yum!) and keeps motivation levels high.
- A book that takes you by the hand to guide you and help you deepen your understanding of Python.

This book is really a labor of love for me and also a huge experiment. I hope you'll enjoy reading it and learn something about Python in the process!

— Dan Bader

1.2 What This Book Will Do for You

My goal for this book is to make you a better—more effective, more knowledgeable, more practical—Python developer. You might be wondering, *How will reading this book help me achieve all that?*

Python Tricks is not a step-by-step Python tutorial. It is not an entry-level Python course. If you're in the beginning stages of learning Python, the book alone won't transform you into a professional Python developer. Reading it will still be beneficial to you, but you need to make sure you're working with some other resources to build up your foundational Python skills.

You'll get the most out of this book if you already have some knowledge of Python, and you want to get to the next level. It will work great for you if you've been coding Python for a while and you're ready to go deeper, to round out your knowledge, and to make your code more Pythonic.

Reading *Python Tricks* will also be great for you if you already have experience with other programming languages and you're looking to get up to speed with Python. You'll discover a ton of practical tips and design patterns that'll make you a more effective and skilled Python coder.

1.3 How to Read This Book

The best way to read *Python Tricks: The Book* is to treat it like a buffet of awesome Python features. Each Python Trick in the book is self-contained, so it's completely okay to jump straight to the ones that look the most interesting. In fact, I would encourage you to do just that.

Of course, you can also read through all the Python Tricks in the order they're laid out in the book. That way you won't miss any of them, and you'll know you've seen it all when you arrive at the final page.

Some of these tricks will be easy to understand right away, and you'll have no trouble incorporating them into your day to day work just by reading the chapter. Other tricks might require a bit more time to crack.

If you're having trouble making a particular trick work in your own programs, it helps to play through each of the code examples in a Python interpreter session.

If that doesn't make things click, then please feel free to reach out to me, so I can help you out and improve the explanation in this book. In the long run, that benefits not just you but all Pythonistas reading this book.

Chapter 2

Sample Chapters

2.1 Object Comparisons: “is” vs “==”

When I was a kid, our neighbors had two twin cats. They looked seemingly identical—the same charcoal fur and the same piercing green eyes. Some personality quirks aside, you couldn’t tell them apart just from looking at them. But of course, they were two different cats, two separate beings, even though they looked exactly the same.

That brings me to the difference in meaning between *equal* and *identical*. And this difference is crucial to understanding how Python’s `is` and `==` comparison operators behave.

The `==` operator compares by checking for *equality*: if these cats were Python objects and we compared them with the `==` operator, we’d get “both cats are equal” as an answer.

The `is` operator, however, compares *identities*: if we compared our cats with the `is` operator, we’d get “these are two different cats” as an answer.

But before I get all tangled up in this ball-of-twine cat analogy, let’s take a look at some real Python code.

First, we’ll create a new list object and name it `a`, and then define another variable (`b`) that points to the same list object:

```
>>> a = [1, 2, 3]
>>> b = a
```

Let’s inspect these two variables. We can see that they point to identical-looking lists:

```
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
```

Because the two list objects look the same, we’ll get the expected result when we compare them for equality by using the `==` operator:

```
>>> a == b
True
```

However, that doesn’t tell us whether `a` and `b` are actually pointing to the same object. Of course, we know they are because we assigned them earlier, but suppose we didn’t know—how might we find out?

The answer is to compare both variables with the `is` operator. This confirms that both variables are in fact pointing to one list object:

```
>>> a is b
True
```

Let’s see what happens when we create an identical copy of our list object. We can do that by calling `list()` on the existing list to create a copy we’ll name `c`:

```
>>> c = list(a)
```

Again you’ll see that the new list we just created looks identical to the list object pointed to by `a` and `b`:

```
>>> c
[1, 2, 3]
```

Now this is where it gets interesting. Let’s compare our list copy `c` with the initial list `a` using the `==` operator. What answer do you expect to see?

```
>>> a == c
True
```

Okay, I hope this was what you expected. What this result tells us is that `c` and `a` have the same contents. They’re considered equal by Python. But are they actually pointing to the same object? Let’s find out with the `is` operator:

```
>>> a is c
False
```

Boom! This is where we get a different result. Python is telling us that `c` and `a` are pointing to two different objects, even though their contents might be the same.

So, to recap, let’s try and break down the difference between `is` and `==` into two short definitions:

- An `is` expression evaluates to `True` if two variables point to the same (identical) object.
- An `==` expression evaluates to `True` if the objects referred to by the variables are equal (have the same contents).

Just remember to think of twin cats (dogs should work, too) whenever you need to decide between using `is` and `==` in Python. If you do that, you’ll be fine.

2.2 Complacent Comma Placement

Here's a handy tip for when you're adding and removing items from a list, dict, or set constant in Python: Just end all of your lines with a comma.

Not sure what I'm talking about? Let me give you a quick example. Imagine you've got this list of names in your code:

```
>>> names = ['Alice', 'Bob', 'Dilbert']
```

Whenever you make a change to this list of names, it'll be hard to tell what was modified by looking at a Git diff, for example. Most source control systems are line-based and have a hard time highlighting multiple changes to a single line.

A quick fix for that is to adopt a code style where you spread out list, dict, or set constants across multiple lines, like so:

```
>>> names = [  
...     'Alice',  
...     'Bob',  
...     'Dilbert'  
... ]
```

That way there's one item per line, making it perfectly clear which one was added, removed, or modified when you view a diff in your source control system. It's a small change but I found it helped me avoid silly mistakes. It also made it easier for my teammates to review my code changes.

Now, there are two editing cases that can still cause some confusion. Whenever you add a new item at the end of a list, or you remove the last item, you'll have to update the comma placement manually to get consistent formatting.

Let's say you'd like to add another name (*Jane*) to that list. If you add *Jane*, you'll need to fix the comma placement after the *Dilbert* line to avoid a nasty error:

```
>>> names = [  
...     'Alice',  
...     'Bob',  
...     'Dilbert' # <- Missing comma!  
...     'Jane'  
]
```

When you inspect the contents of that list, brace yourself for a surprise:

```
>>> names  
['Alice', 'Bob', 'DilbertJane']
```

As you can see, Python *merged* the strings *Dilbert* and *Jane* into *DilbertJane*. This so-called “string literal concatenation” is intentional and documented behavior. And it's also a fantastic way to shoot yourself in the foot by introducing hard-to-catch bugs into your programs:

“Multiple adjacent string or bytes literals (delimited by whitespace), possibly using different quoting conventions, are allowed, and their meaning is the same as their concatenation.”¹

Still, string literal concatenation is a useful feature in some cases. For example, you can use it to reduce the number of backslashes needed to split long string constants across multiple lines:

¹cf. Python Docs: “[String literal concatenation](#)”

```
my_str = ('This is a super long string constant '  
          'spread out across multiple lines. '  
          'And look, no backslash characters needed!')
```

On the other hand, we've just seen how the same feature can quickly turn into a liability. Now, how do we fix this situation?

Adding the missing comma after *Dilbert* prevents the two strings from getting merged into one:

```
>>> names = [  
...     'Alice',  
...     'Bob',  
...     'Dilbert',  
...     'Jane'  
]
```

But now we've come full circle and returned to the original problem. I had to modify two lines in order to add a new name to the list. This makes it harder to see what was modified in the Git diff again... Did someone add a new name? Did someone change Dilbert's name?

Luckily, Python's syntax allows for some leeway to solve this comma placement issue once and for all. You just need to train yourself to adopt a code style that avoids it in the first place. Let me show you how.

In Python, you can place a comma after every item in a list, dict, or set constant, including the last item. That way, you can just remember to always end your lines with a comma and thus avoid the comma placement juggling that would otherwise be required.

Here's what the final example looks like:

```
>>> names = [  
...     'Alice',
```

```
...     'Bob',  
...     'Dilbert',  
... ]
```

Did you spot the comma after *Dilbert*? That'll make it easy to add or remove new items without having to update the comma placement. It keeps your lines consistent, your source control diffs clean, and your code reviewers happy. Hey, sometimes the magic is in the little things, right?

Key Takeaways

- Smart formatting and comma placement can make your list, dict, or set constants easier to maintain.
- Python's string literal concatenation feature can work to your benefit, or introduce hard-to-catch bugs.

This is a sample from “Python Tricks: The Book”

The full version of the book includes many more Python Tricks that will teach you the depths and quirks of Python with fun and easy to understand examples and explanations.

If you enjoyed the sample chapters you can purchase a full version of the book at realpython.com/pytricks-book.
