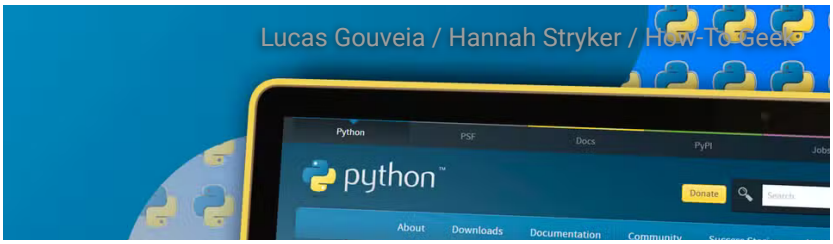
How to Write Code the Pythonic Way (With 6 Examples)

By Zunaid Ali — 5 days ago



The Python programming language has a lot to offer. Whether you're new to programming or just new to Python, you'll fin
wr

egant, readable ways to handle coding.

What Does Pythonic Mean?

Every programming language has its own quirks and conventions—ways of doing things that feel natural in that language. In Python, there's a particular style and philosophy that encourages writing code that's clean, readable, and elegant. When developers say code is Pythonic, they mean that it follows these principles and makes the most of what Python has to offer.

Instead of just writing code that works, Python encourages you to write code that's also beautiful. Code that's easy to understand at a glance, even for someone who didn't write it. Non-Pythonic code often looks like it was written in another language and just translated into Python. Pythonic code embraces Python's strengths. This includes things like readable syntax, powerful built-ins, and expressive one-liners.

You'll often hear Python developers refer to something called the Zen of Python. It's a set of guiding principles that influence the way Python is written. It's also built in as a fun Easter egg in the language.

Why Should You Write Code in the Pythonic Way?

Mark LoProto / How-To Geek

So you know what Pythonic code is. But why go out of your way to write it? The answer is simple. It's because it makes your life easier. Writing Pythonic code isn’t just about following some unwritten style rules or showing off clever tricks. It’s about making your code more readable, more maintainable, and often more efficient.

Python is packed with features designed to reduce boilerplate and repetitive patterns. Once you learn the common idioms, you'll find yourself writing less code to do more. Learning to write Pythonic code helps you "think in Python." You'll start to recognize patterns and use the standard library more effectively. Moreover, you'll feel more comfortable working with other people's Python code.

6 Examples of Pythonic Code

Let's dive into some cool examples of Pythonic code that will change how you code in Python.

String Reverse

Let’s start with a classic problem: reversing a string. In many languages, you’d write a loop to iterate through each character to build the reversed string.

```
input_string = 'hello'
reversed_string = ''
for char in input_string:
   reversed_string = char + reversed_string
print(reversed_string)
```

This works fine. Each character gets prepended to build the reversed string. But it's a bit clunky. Here's the Pythonic version:

```
reversed_string = input_string[::-1]
```

Just one line. This uses Python’s slice syntax. [::-1] means “take the whole string

but step backwards.”

Checking for Membership

Let’s say you want to check if a particular item exists in a list. In many languages, this usually means writing a loop and some conditionals.

```
fruits = ['strawberry', 'orange','apple', 'mango']
found = False
for fruit in fruits:
   if fruit == 'apple':
       found = True
       break
print(found)
```

It’s a lot of code just to find out if "apple" is in the list. Here's how to do it in Python:


```
found = 'apple' in fruits
```

This one-liner reads almost like plain English. It eliminates the need for manual flags and reduces the chance of introducing bugs in your code.

Checking Multiple Conditions with `any()` and `all()`

Sometimes you want to check if any or all items in a list meet a certain condition. In many languages, you'd use a loop with flags or counters to do this.

```
has_negative = False
for num in numbers:
   if num < 0:
       has_negative = True
       break
```

This is clear enough, but it adds extra lines and variables. You can do this with the `any()` function in Python.

```
has_negative = any(num < 0 for num in numbers)
```

And if you want to make sure all numbers are positive:

```
all_positive = all(num > 0 for num in numbers)
```

These functions work by evaluating a condition across all elements and stopping as soon as the result is known.

Combining Strings with `join()`

If you've ever needed to build a sentence or combine characters or words from a list, your first instinct might be to loop through them and add each one manually, like this:

```
sentence = ''
for word in words:
   sentence += word + ' '
```

This works, but it's inefficient, especially

with large lists. Using Python's join() function, you can simply do this:

```
sentence = ' '.join(words)
```

It’s cleaner, faster, and more expressive. You just specify the separator (' ' in this case) and call `.join()` on it, passing in the list of strings. Make sure the list contains only strings because join() won’t work with numbers or other data types unless you convert them first.

Counting Items with `collections.Counter`

Counting how often each item appears in a list or string is a task that comes up all the time. This is how you'd typically approach the problem in most languages:

check if the key exists, then increment it:

```
counts = {}
for item in items:
   if item in counts:
       counts[item] += 1
   else:
       counts[item] = 1
```

Using the collections module, you can do the same with much less code.

```
from collections import Counter
counts = Counter(items)
```

Just one line, and it does everything for you. It returns a dictionary-like object where keys are the items and values are their counts. Counter works on any iterable, not just lists. You can use it on strings, tuples, or even the output of a generator.

In-Place Variable Swapping with Tuple Unpacking

In many programming languages, swapping values between two variables requires a temporary variable.

```
temp = a
a = b
b = temp
```

This is fine, but it's extra work and feels outdated in Python. Here's the Pythonic method:

```
a, b = b, a
```

No temp variable. No clutter. This uses a concept called tuple unpacking, and swapping variables is just the beginning. You can also use it to unpack multiple values at a time.

```
name, age, country = ['Alice', 30, 'Canada']
```

No need to access each element by index. This works with tuples, lists, or any iterable of the right length.

Look Out for Readability

While writing Pythonic code is often a sign of clean, elegant programming, it can sometimes go too far. Some Pythonic patterns, especially when used excessively or in the wrong context, can make your code harder to read, not easier. One of the core values of Python is its readability. In fact, one of the Zen of Python principles says, "Readability counts." So yes, using idiomatic Python is good, but not at the cost of turning your code into a puzzle.

List comprehensions are a good example

of this. They're concise, fast, and expressive. However, when you start nesting them or adding too many conditions, they can become hard to follow. Let's look at an example.


```
filtered = []
for user in users:
   if user.is_active and user.age > 18:
       filtered.append(user.name)
```

This is the version using list comprehension:

```
filtered = [user.name for user in users if user.is_active and user.age > 18]
```

While the list comprehension is still OK here, it quickly becomes unreadable if more logic is added (like nested loops or

multiple conditions.) If your comprehension is longer than one line or takes a second glance to understand, it might be better to stick with a loop.

There are so many reasons to learn Python, and its elegant way of writing code is just one of them. Once you get the hang of it, you can explore other tricks, such as using it as a calculator or to understand your mobile's battery life.

Programming

Follow | Like | Share

*Readers like you help support How-To Geek. When you make a purchase using links on our site, we may earn an affiliate commission. **Read More**.*

THREAD 9

We want to hear from you! Share your opinions in the thread below and remember to keep it respectful.

SMuggi

Images

Please respect our community guidelines. No links, inappropriate language, or spam.

Sort by: Popular

Ozan

2025-06-06 15:29:38

Last example of long list comprehension is fine, but I'd use newline before the if statements. That makes it much easier to read. Newlines are valid inside brackets.

filtered = [user.name for user in users
if user.is_active and user.age > 18]

I can't use indentation in this comment section. Second line was supposed to be in line with the first.

(Edited)

2 Copy

mark

2025-06-08 10:03:32

I think the negative-stride mechanism impinges somewhat on the readability goal. Definitely correct and efficient Python, but I suspect that most of us don't use slices all the time, and even then, mainly not with strides.

And I guess the least surprise-way to reverse a string in Python really should be reversed(). But it's not, of course, because of the way an iterator on a string explodes it to a list of chars. Which I find somewhat unpythonic :)

1 1 Copy

Roc

2025-05-09 11:19:07

I don't think I've ever needed to reverse a string since 1979... Maybe once in a class assignment.

(edited)

1 Copy

KevinPa

2025-06-10 20:59:32

This reminds me of using APL (look it up) 40-some years ago. You could do amazing things with a single line of code. Nobody except the author could tell what was supposed to do by reading it, and a lot of authors couldn't figure out their own code a couple weeks after writing it.

1 Copy

Neilw20

2025-06-10 14:01:51

As a c programmer for 40 years, I hate python. Cryptic crap.

1 Copy

Vitor

2025-06-09 20:39:49

I code use AI in Cursor....

Copy

Shane

2025-06-07 23:52:25

Nothing against Python, it is a cool language... but many other languages also have their cool shortcuts too. The plethora of examples the author used are avoided in multiple languages, not just python.

1 Copy

VADLA VINAY

2025-06-08 07:47:00

I have nothing against Python — it's definitely a great language. But let's not forget that many other languages have their own elegant shortcuts and features too. In fact, the patterns the author highlighted are commonly avoided across several languages, not just in Python.

1 Copy

mark

2025-06-08 18:52:17

Is this just a paraphrase of the previous comment? Perhaps even merely AI generated?

Copy

RECOMMENDED

Windows Vista

Windows Vista Was a Mess, but It Did One Thing Right

I dug it then. I dig it now.

6 days ago

23

iPad

8 Ways I'm Putting My Old iPad to Good Use

Your outdated iPad can still be put to good use.

7 days ago

13

Open Source

7 Open-Source Windows Apps I Can't Live Without

Not every program needs to be proprietary.

4 days ago

20

Audio

Why Vinyl Doesn't Actually Sound Better

Who likes the pop, hiss, and crackle?

4 days ago

82

Streaming

I've Watched Every Star Trek TNG Episode—These Are the 10 Best

Now with 10% more Riker's beard.

4 days ago

20

Linux

7 Things I Wish I Knew Before Running a Pi-hole

Don't be like me, make sure to prepare for these things before deploying Pi-hole the first time.

4 days ago

11