Aug 14, 2018 | in Tutorials 🔖 python 📝 Edit

Python Mutable Defaults Are The Source of All Evil

How to prevent a common Python mistake that can lead to horrible bugs and waste everyone's time.

Green python on brown tree. @tirzavandijk, unsplash.com

Today, I wanted to share a quick **life-saving advice** about a common mistake Python developers still tend to make.

TL;DR

Do not use mutable default arguments in Python, unless you have a REALLY good reason to do so.

Why? Because it can lead to all sorts of **nasty and horrible bugs**, give you **headaches** and **waste everyone's time**.

Instead, default to `None` and assign the mutable value inside the function.

Story time

This error caught me a few times in my early developer journey.

- When I was just starting making real projects with Python (probably 5-6 years ago), I remember stumbling upon a very strange bug. A list would grow bigger than expected when a function was called multiple times, which would cause strange errors.
- Once in college, we were assigned an algorithm development project and the instructor sent us a program that unit-tested our code. At one point, I was absolutely convinced that my code was correct, yet the tests kept failing.

In the first case, the young-developer-learning-how-to-debug me was getting angry because he had no idea why that list was growing too big, and he spent **hours** trying to fix this issue.

In the second case, other fellow students also encountered the problem, and the instructor had no clue either, so **everyone's time was wasted**.

In both cases, time and effort could have been saved if we had known about this common mistake.

What were we doing wrong?

It turns out that in both cases, **an empty list was used as a default argument to a function**. Yep, just that. Something like:

```
def compute_patterns(inputs=[]):
    inputs.append("some stuff")
    patterns = ["a list based on"] + inputs
    return patterns
```

Try it out yourself: if you run this function multiple times, you'll get different results!

```
>>> compute_patterns()
['a list based on', 'some stuff']  # Expected
>>> compute_patterns()
['a list based on', 'some stuff', 'some stuff']  # Woops!
```

Although it doesn't look like much, **`inputs=[]` was the naught boy causing of this mess**.

The problem

In Python, when passing a mutable value as a default argument in a function, **the default argument is mutated anytime that value is mutated**.

Here, "mutable value" refers to anything such as a list, a dictionnary or even a class instance.

What happened, exactly?

It is not straight-forward to see why the above fact can be a problem.

Here's what happens in detail using another example:

```
def append(element, seq=[]):
    seq.append(element)
    return seq
```

```
>>> append(1)  # seq is assigned to []
[1]  # This returns a reference the *same* list as the default for `seq`
>>> append(2)  # => `seq` is now given [1] as a default!
[1, 2]  # WTFs and headaches start here…
```

As you can see, Python "retains" the default value and ties it to the function in some way. Since it is mutated inside the function, it is also mutated as a default. In the end, **we use a different default every time the function is called** — duh!

The solution

Long story short, the solution is simple.

Use `None` as a default and assign the mutable value inside the function.

So instead, do this:

```
# 👇
def append(element, seq=None):
    if seq is None:  # 👍
        seq = []
    seq.append(element)
    return seq



>>> append(1)  # `seq` is assigned to []
[1]
>>> append(2)  # `seq` is assigned to [] again!
[2]  # Yay!
```

This is actually a very common pattern in Python; I find myself writing `if some_var is None: some_var = default_value` literally dozens of time in every project.

It's so common that there is a Python Enhancement Proposal (PEP) currently in the works (PEP 505 - None-aware operators) that would, among other things, allow to simplify the above code and simply write:

```
def append(element, seq=None):
    seq ??= []  # ✨
    seq.append(element)
    return seq
```

The PEP is still a draft, but I just wanted to mention it because I really hope none-aware operators soon become a thing in Python! 🔥

Lessons learned

There you go! Hopefully you'll never see yourself using mutable defaults in Python code anymore (except if you want to mess with everyone's nerves).

If you see someone else using them, spread the tip and save their lives too. 🙏