Snippets with tipps and tricks for Python

Small Tips & Tricks for Python development. I've been programming a lot with Python until 2017 but took a long break. Now I'm more and more stepping in again because I'm using it at work. In this article I gather useful snippets I find along the way.

Contents

1 Also see in English

- 2017: Strengths and weaknesses of Python
- 2017: minimal Python script (a useful skeleton to start with)
- 2015: Real Life Infocalypse (Decentralized DVCS hosting in the Darknet)
- 2015: Freenet Communication Primitives: Part 1, Files and Sites
- 2014: Memory requirement of Python datastructures: numpy array, list of floats and inner array
- 2013: Equal-Area Map Projections with Basemap and matplotlib/pylab
- 2012: Read your python module documentation from emacs
- 2012: Minimal example for literate programming with noweb in emacs org-mode
- 2008: With Python from the Shadows (a song `:-)`)
- 2007: Python for beginning programmers

2 Also see in German

- 2016: Warum Python 3? (Kurzfassung, was Python 3 verbesserte, damit ich das in meiner Frustration über seine Volatilität nicht vergesse)
- 2012: Iteratoren als Filter
- 2009: Spiele programmieren (Quellen)
- 2008: Blob Valentine (ein kleines Geschenk)
- 2008: Geschwindigkeitstest - eine Liste summieren, selbstgeschrieben oder mit sum()
- 2008: Blob Schwarm

3 Doctests in Django

I consider doctests to be one of the most elegant ways to test beautifully self-contained functions with easy to understand input.

But using them in Django is not as straightforward as I expected since the documentation just says "we use unittest".

That said, using the doctest unittest API just works.

If you have a file at `app/utils/strings.py` with a function:

```
def string_postfix_to_number(versioned_name):
    """Turn a string with a number suffix into the number.

    >>> string_postfix_to_number("ABC-05")
    5
    >>> string_postfix_to_number("ABC-09")
    9
    >>> string_postfix_to_number("ABC-99")
    99
    """
    return int(versioned_name.split("-")[-1])
```

and your unit test is at `app/tests/unit/tests_models.py`, just add to `tests_models.py`:

```
from django.test import TestCase

import unittest
import doctest
from app.utils import strings
```

```
def load_tests(loader, tests, ignore):
    tests.addTests(doctest.DocTestSuite(strings))
    return tests

class SomeUnitTest(TestCase):
    pass
```

Now `python3 manage.py test app` will execute your doctests.

If you have a logic that needs no setup — i.e. just takes primitive data structures — doctests are the most convenient way to quickly add some tests. Just start writing your tests at the top of your function. I often do it before writing the function body itself and that way Test Driven Development (TDD) becomes a breeze.

I like these so much, that I created doctests for Guile Scheme.

If you need setup or your own classes or you want to test the interaction between functions, an external test like the `SomeUnitTest` in this example is often better, though.

There's a lot of support in Django for those more complex tests.

Thanks to Adam Johnson for pointing me to the right docs when I ranted on Mastodon about removed doctest support.

[2025-03-28 Fr]