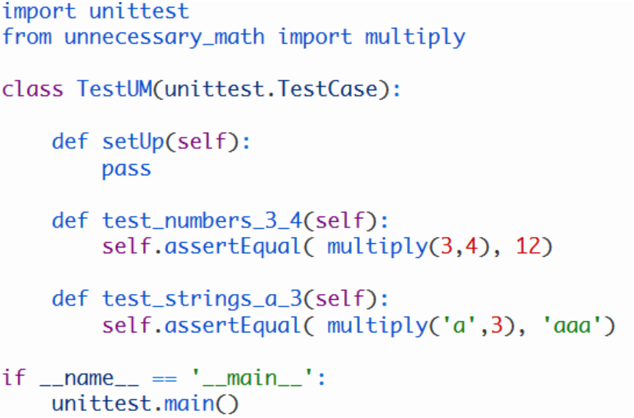
Testing tools in Python

Benedikt Kauf

16.04.2019

pytest

Standard tool: *unittest*

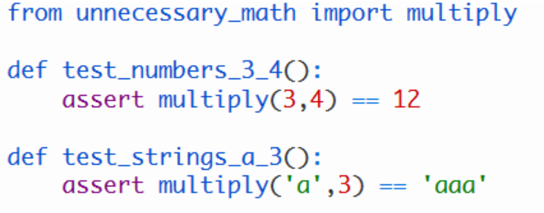


```python
import unittest
from unnecessary_math import multiply

class TestUM(unittest.TestCase):

    def setUp(self):
        pass

    def test_numbers_3_4(self):
        self.assertEqual( multiply(3,4), 12)

    def test_strings_a_3(self):
        self.assertEqual( multiply('a',3), 'aaa')

if __name__ == '__main__':
    unittest.main()
```

Inconveniences of using *unittest*

- Need to collect all tests in a test class that inherits from *unittest*
- Requires specific *assert*-statements
- Test files have to be run separately
- Test output can become unnecessarily convoluted

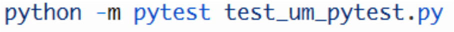
Solution: *pytest*



```
from unnecessary_math import multiply

def test_numbers_3_4():
    assert multiply(3,4) == 12

def test_strings_a_3():
    assert multiply('a',3) == 'aaa'
```

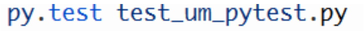
How to run tests in *pytest*

Test files can either be run separately by running



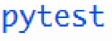
```
python -m pytest test_um_pytest.py
```

or



```
py.test test_um_pytest.py
```

or jointly, by accessing the directory with the test files in your shell and simply typing



```
pytest
```

in your command line.

Reasons for *pytest* over *nose*

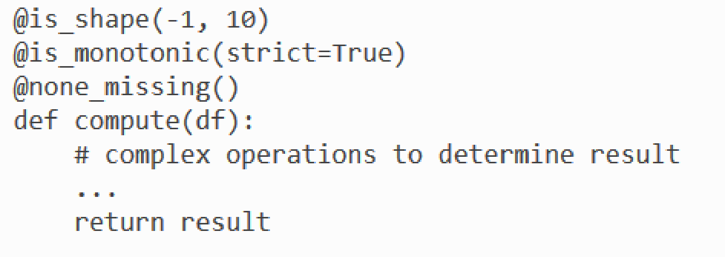
- Test output is more clearcut in *pytest*
- Contrary to *nose*, *pytest* is still being actively developed
- *nose* requires a function import in order to serve factual assert statements, e. g. *assert_contains(x, y)* or *assert_is(a, b)*

Engarde

Why Engarde?

- Data are messy
- Often we have assumptions that should be invariant across updates to your dataset, e. g. that there are no missing values or that all values are within a certain range
- Engarde provides a lightweight way to check the correctness of these assumptions by using decorators

Basic setup



```
@is_shape(-1, 10)
@is_monotonic(strict=True)
@none_missing()
def compute(df):
    # complex operations to determine result
    ...
    return result
```

Assumptions that can be checked by Engarde (among others)

- There are no missing values
- A dataframe only contains variables of certain data types
- All values are within 3 standard deviations of the mean
- Some/all values adhere to a self-programmed criterion

Hypothesis

Hypothesis vs. normal unit tests

What unit tests usually do:

1. Set up some data
2. Perform some operations on the data
3. Assert something about the result

What Hypothesis does:

1. For all data matching some specification.
2. Perform some operations on the data
3. Assert something about the result

→ property-based testing

Hypothesis

In other words:

- You specify some assumptions that the data is supposed to adhere to, e. g. the data should only consist of integers
- Hypothesis tests your code for a large number of data specifications that fulfill the prespecified assumptions including a lot of corner cases
- If it finds a counterexample, it will try to simplify the example as much as possible and finally return the simplified counterexample

→ particularly useful for complex code which might be prone to missing some special cases