

# Unit Testing

Sebastian Raschka  
January 8, 2014



<http://xkcd.com/1301/>

# What is unit testing?

"unit" of code:

module, class, function, data file

in isolation!

# What is unit testing?

a "unit":

- 1) **fixture** (e.g., function)
- 2) **action** (e.g., invoking function with particular input)
- 3) **expected result** (e.g., return value of a function)
- 4) **actual result** (e.g., actual return value of a function)
- 5) **report** (e.g., success or failure)

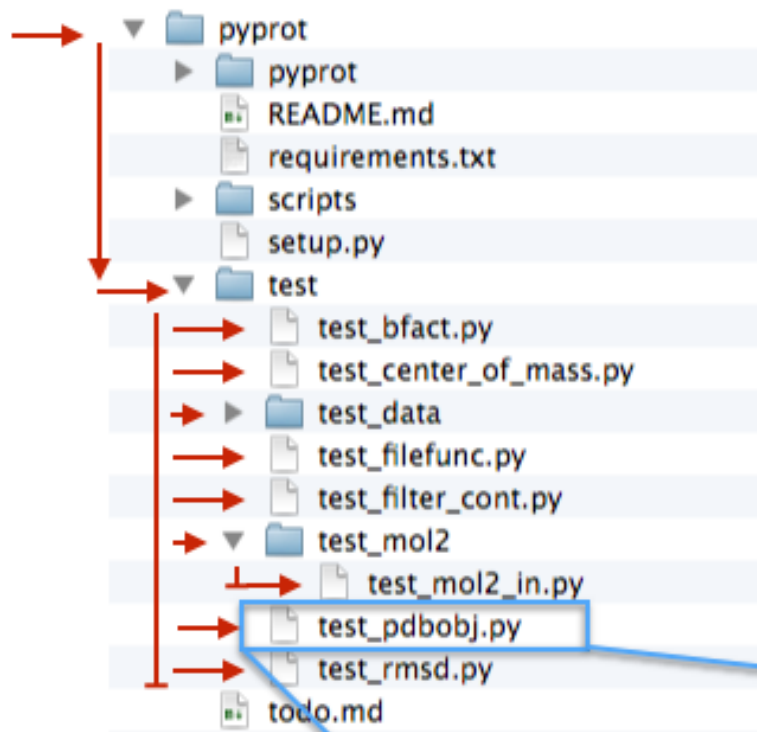
# Why unit testing?

- testing functionality (incl. edge cases)
- alter your code and make sure that you didn't break anything
- debugging
- trust your code
- save time in the long run
- trust code from collaborators
- credibility for publication

# unit testing frameworks in python

- unittest
- nose
- py.test

nose descending the  
directory tree looking for  
prefix "**test**"

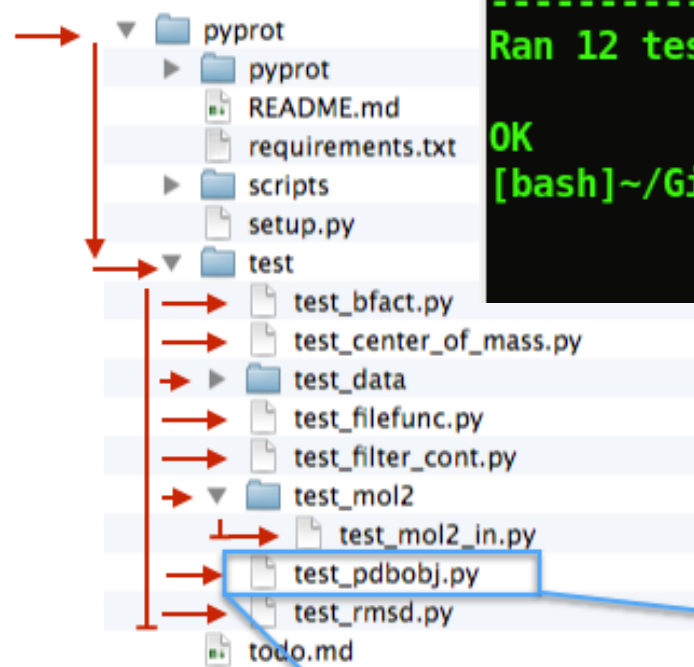


```
test_pdbobj.py (~/Github/pyprot/test) - VIM
import pyprot.pdb as ppdb

pdb1 = ppdb.PdbObj("../test/test_data/3EII.pdb")
pdb2 = ppdb.PdbObj("../test/test_data/small_3EII.pdb")

def test_constructor():
    assert len(pdb1.cont) == 2147
    assert len(pdb1.atom) == 1330
    assert len(pdb1.hetatm) == 151
    assert len(pdb1.conect) == 54

def test_calpha():
    atom = [
        'ATOM      2  CA  SER A   2      3.259  54.783 -0.368  1.00 52.54      C',
        'ATOM      8  CA  PHE A   3      4.261  51.413  1.102  1.00 48.73      C',
        'ATOM     19  CA  SER A   4      4.593  49.745 -2.323  1.00 47.00      C',
        'ATOM     25  CA  ASN A   5      7.351  52.145 -3.480  1.00 41.42      C',
        'ATOM     33  CA  VAL A   6      9.636  51.959 -0.457  1.00 32.41      C'
    ]
    assert atom == pdb2.calpha()
```



```
Terminal
File Edit View Search Terminal Help

[bash]~/Gits/Github/pyprot >nosetests
.....
-----
Ran 12 tests in 0.143s

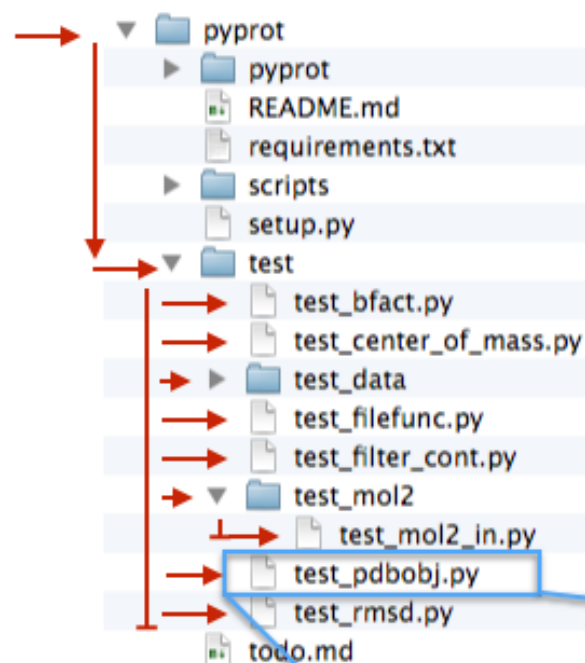
OK
[bash]~/Gits/Github/pyprot >
```

```
test_pdbobj.py (~/Github/pyprot/test) - VIM
import pyprot.pdb as ppdb

pdb1 = ppdb.PdbObj("../test/test_data/3EII.pdb")
pdb2 = ppdb.PdbObj("../test/test_data/small_3EII.pdb")

def test_constructor():
    assert len(pdb1.cont) == 2147
    assert len(pdb1.atom) == 1330
    assert len(pdb1.hetatm) == 151
    assert len(pdb1.conect) == 54

def test_calpha():
    atom = [
        'ATOM      2  CA  SER A   2         3.259  54.783 -0.368  1.00 52.54      C',
        'ATOM      8  CA  PHE A   3         4.261  51.413  1.102  1.00 48.73      C',
        'ATOM     19  CA  SER A   4         4.593  49.745 -2.323  1.00 47.00      C',
        'ATOM     25  CA  ASN A   5         7.351  52.145 -3.480  1.00 41.42      C',
        'ATOM     33  CA  VAL A   6         9.636  51.959 -0.457  1.00 32.41      C'
    ]
    assert atom == pdb2.calpha()
```



```
Terminal
File Edit View Search Terminal Help

[bash]~/Gits/Github/pyprot >nosetests -v
test_bfact.test_get_bfactor ... ok
test_bfact.test_median_bfactor ... ok
test_bfact.test_mean_bfactor ... ok
test_center_of_mass.test_center_of_mass ... ok
test_filefunc.test_open_file ... ok
test_filter_cont.test_filter_column_match ... ok
test_filter_cont.test_filter_col_match_exclude ... ok
test_mol2_in.test_multi_mol2list ... ok
test_pdbobj.test_constructor ... ok
test_pdbobj.test_calpha ... ok
test_pdbobj.test_main_chain ... ok
test_rmsd.test_rmsd ... ok

-----
Ran 12 tests in 0.128s

OK
[bash]~/Gits/Github/pyprot >
```

```
test_pdbobj.py (~/Github/pyprot/test) - VIM
import pyprot.pdb as ppdb

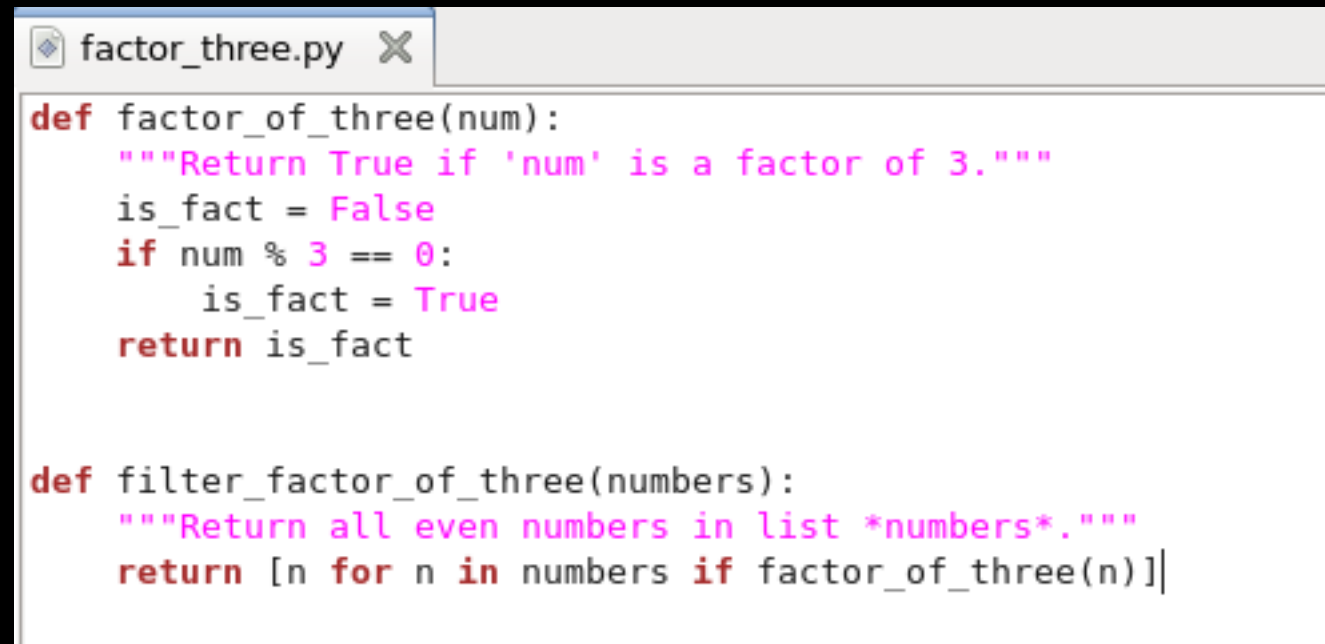
pdb1 = ppdb.PdbObj("../test/test_data/3EII.pdb")
pdb2 = ppdb.PdbObj("../test/test_data/small_3EII.pdb")

def test_constructor():
    assert len(pdb1.cont) == 2147
    assert len(pdb1.atom) == 1330
    assert len(pdb1.hetatm) == 151
    assert len(pdb1.conect) == 54

def test_calpha():
    atom = [
        'ATOM      2  CA  SER A   2         3.259  54.783  -0.368  1.00  52.54      C',
        'ATOM      8  CA  PHE A   3         4.261  51.413   1.102  1.00  48.73      C',
        'ATOM     19  CA  SER A   4         4.593  49.745  -2.323  1.00  47.00      C',
        'ATOM     25  CA  ASN A   5         7.351  52.145  -3.480  1.00  41.42      C',
        'ATOM     33  CA  VAL A   6         9.636  51.959  -0.457  1.00  32.41      C'
    ]
    assert atom == pdb2.calpha()
```



# Example: Testing a simple function



```
factor_three.py X
def factor_of_three(num):
    """Return True if 'num' is a factor of 3."""
    is_fact = False
    if num % 3 == 0:
        is_fact = True
    return is_fact

def filter_factor_of_three(numbers):
    """Return all even numbers in list *numbers*."""
    return [n for n in numbers if factor_of_three(n)]
```

# Example: Unit test for our function

```
factor_three.py X
def factor_of_three(num):
    """Return True if 'num' is a factor of 3."""
    is_fact = False
    if num % 3 == 0:
        is_fact = True
    return is_fact

def filter_factor_of_three(numbers):
    """Return all even numbers in list *numbers*."""
    return [n for n in numbers if factor_of_three(n)]
```

```
factor_three.py X *test_factor_three.py X
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
```

# Example: Unit test for our function

```
factor_three.py X
def factor_of_three(num):
    """Return True if 'num' is a factor of 3."""
    is_fact = False
    if num % 3 == 0:
        is_fact = True
    return is_fact

def filter_factor_of_three(numbers):
    """Return all even numbers in list *numbers*."""
    return [n for n in numbers if factor_of_three(n)]
```

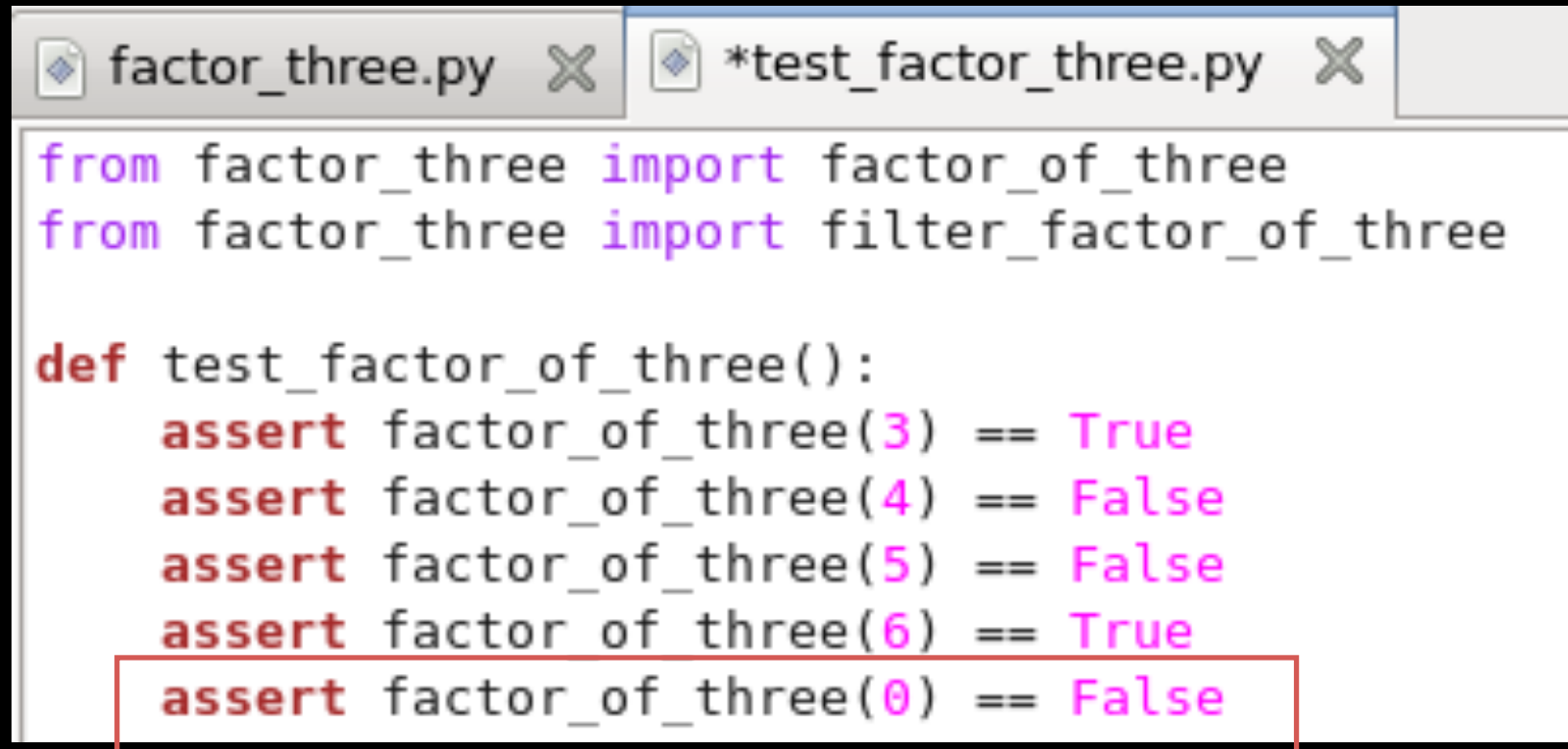
```
factor_three.py X *test_factor_three.py X
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
```

```
Terminal
File Edit View Search Terminal Help
[bash]~/Desktop >nosetests
.
-----
Ran 1 test in 0.009s

OK
[bash]~/Desktop >█
```

# Example: Testing edge cases

A screenshot of a code editor window with two tabs: 'factor\_three.py' and '\*test\_factor\_three.py'. The active tab is '\*test\_factor\_three.py', which contains Python code for testing a function. The code imports 'factor\_of\_three' and 'filter\_factor\_of\_three' from 'factor\_three'. It then defines a function 'test\_factor\_of\_three()' with five assertions. The last assertion, 'assert factor\_of\_three(0) == False', is highlighted with a red rectangular box, indicating it is an edge case test.

```
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
    assert factor_of_three(0) == False
```

# Example: Testing edge cases

```
factor_three.py  *test_factor_three.py
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
    assert factor_of_three(0) == False
```

```
[bash]~/Desktop >nosetests
```

```
F
```

```
=====
FAIL: test_factor_three.test_factor_of_three
-----
```

```
Traceback (most recent call last):
```

```
  File "/soft/linux64/python/3.3.3--GCC-4.4.7/lib/python3.3/site-packages/nose-0.3.0-py3.3.egg/nose/case.py", line 198, in runTest
    self.test(*self.arg)
```

```
  File "/home/raschkas/Desktop/test_factor_three.py", line 9, in test_factor_of_three
    assert factor_of_three(0) == False
```

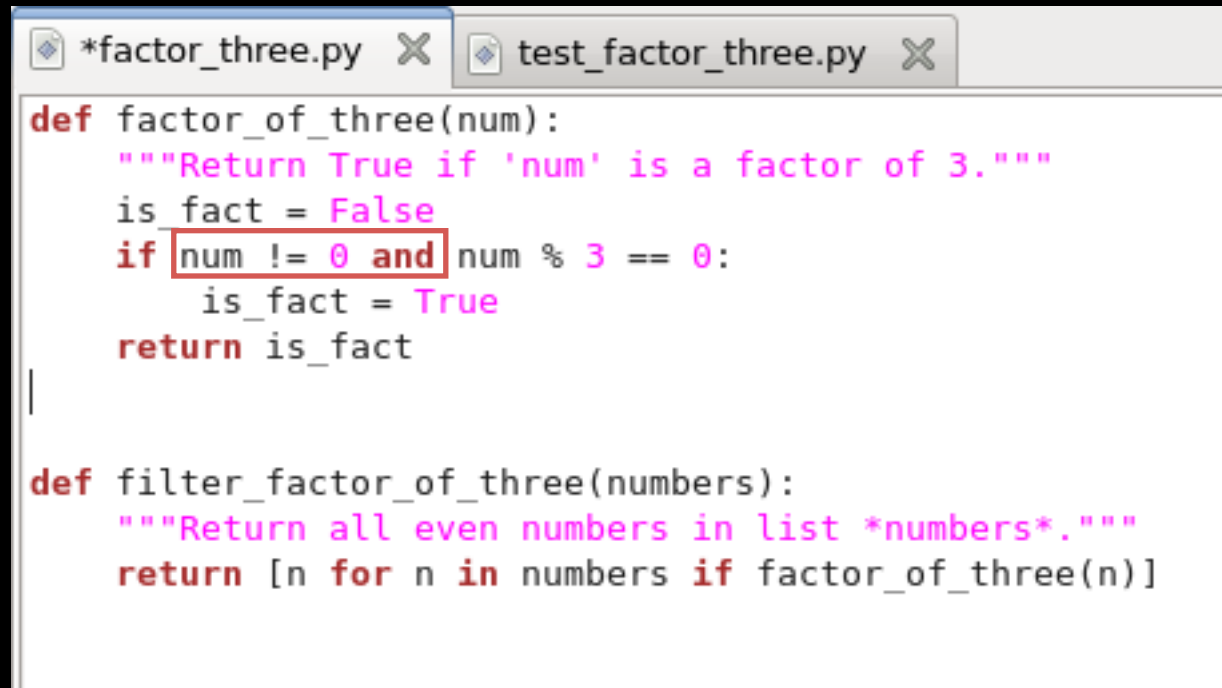
```
AssertionError
```

```
-----
Ran 1 test in 0.010s
```

```
FAILED (failures=1)
```

```
[bash]~/Desktop >
```

# Example: Fixing the code



```
def factor_of_three(num):  
    """Return True if 'num' is a factor of 3."""  
    is_fact = False  
    if num != 0 and num % 3 == 0:  
        is_fact = True  
    return is_fact  
  
def filter_factor_of_three(numbers):  
    """Return all even numbers in list *numbers*."""  
    return [n for n in numbers if factor_of_three(n)]
```

# Example: Fixing the code

```
*factor_three.py X test_factor_three.py X
def factor_of_three(num):
    """Return True if 'num' is a factor of 3."""
    is_fact = False
    if num != 0 and num % 3 == 0:
        is_fact = True
    return is_fact

def filter_factor_of_three(numbers):
    """Return all even numbers in list *numbers*."""
    return [n for n in numbers if factor_of_three(n)]
```

```
[bash]~/Desktop >nosetests
```

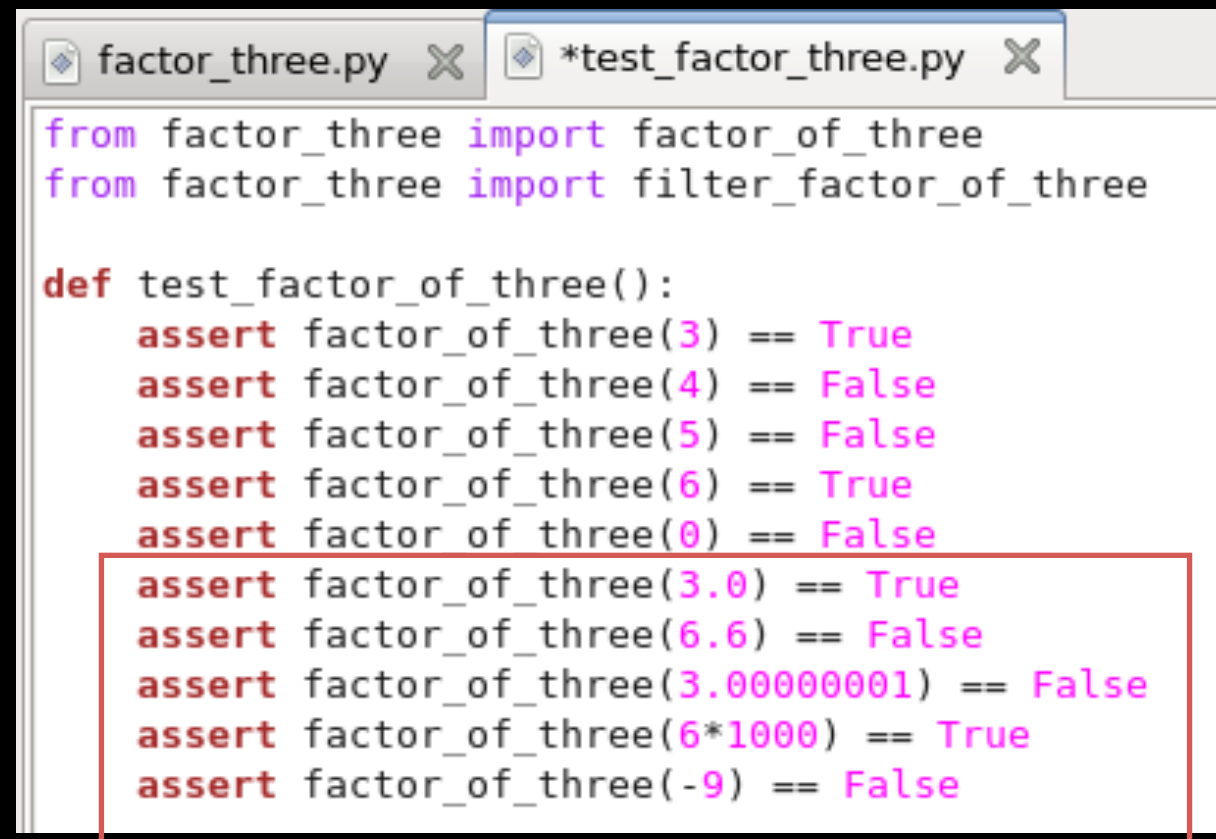
```
.
```

```
-----
Ran 1 test in 0.009s
```

```
OK
```

```
[bash]~/Desktop >
```

# Example: Testing more edge cases



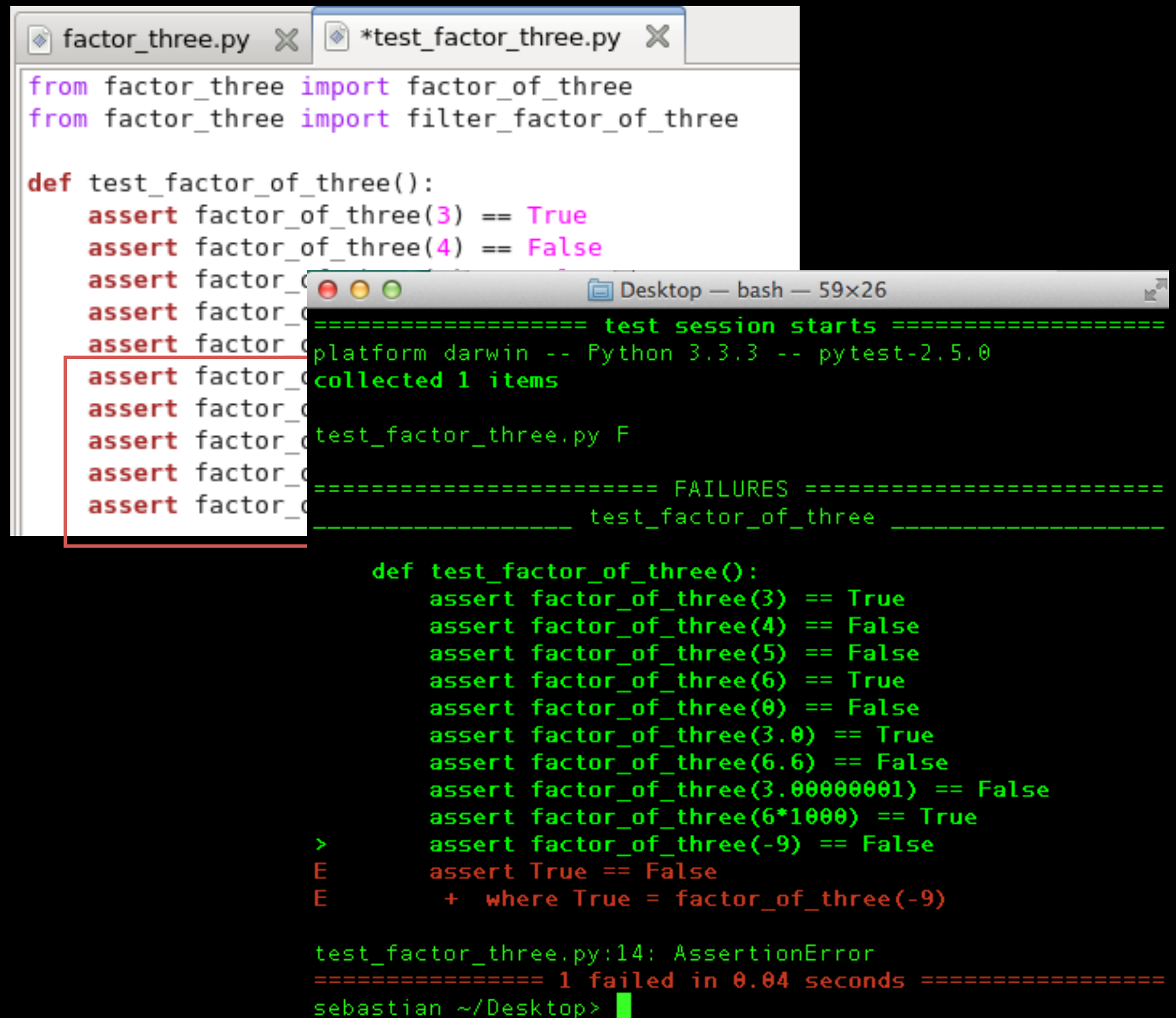
```
factor_three.py X *test_factor_three.py X
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
    assert factor_of_three(0) == False
    assert factor_of_three(3.0) == True
    assert factor_of_three(6.6) == False
    assert factor_of_three(3.00000001) == False
    assert factor_of_three(6*1000) == True
    assert factor_of_three(-9) == False
```

The image shows a code editor with two tabs: 'factor\_three.py' and '\*test\_factor\_three.py'. The active tab is '\*test\_factor\_three.py', which contains a Python function 'test\_factor\_of\_three()'. This function uses 'assert' statements to verify the behavior of 'factor\_of\_three()' for various inputs. The inputs include integers (3, 4, 5, 6, 0), a float (3.0), a non-integer float (6.6), a float very close to an integer (3.00000001), a large integer (6\*1000), and a negative integer (-9). The expected results are True or False. A red rectangular box highlights the last five lines of code, which represent the 'edge cases' mentioned in the title: 3.0, 6.6, 3.00000001, 6\*1000, and -9.



# Example: Testing more edge cases



The image shows a code editor with two tabs: `factor_three.py` and `*test_factor_three.py`. The `test_factor_three.py` file contains a test function `test_factor_of_three()` with several `assert` statements. A red box highlights the last five assertions, which are all `assert factor_of_three(0) == False`. Below the code editor, a terminal window titled "Desktop — bash — 59x26" shows the output of running `pytest`. The output indicates that the test session started on a Darwin platform using Python 3.3.3 and pytest-2.5.0. It collected 1 item and ran the test `test_factor_of_three`, which failed. The failure is an `AssertionError` at line 14, where the assertion `assert True == False` failed. The terminal output also shows the traceback, indicating that the failure occurred in the `test_factor_of_three` function.

```
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
    assert factor_of_three(0) == False
    assert factor_of_three(3.0) == True
    assert factor_of_three(6.6) == False
    assert factor_of_three(3.00000001) == False
    assert factor_of_three(6*1000) == True
    assert factor_of_three(-9) == False
    assert True == False
    + where True = factor_of_three(-9)

test_factor_of_three.py:14: AssertionError
===== 1 failed in 0.04 seconds =====
sebastian ~/Desktop>
```

# Example: Fixing the code again

```
*factor_three.py X test_factor_three.py X
def factor_of_three(num):
    """Return True if 'num' is a factor of 3."""
    is_fact = False
    if num != 0 and num % 3 == 0:
        is_fact = True
    return is_fact

def filter_factor_of_three(numbers):
    """Return all even numbers in list *numbers*."""
    return [n for n in numbers if factor_of_three(n)]
```

```
*factor_three.py X test_factor_three.py X
def factor_of_three(num):
    """Return True if 'num' is a factor of 3."""
    is_fact = False
    if num > 0 and num % 3 == 0:
        is_fact = True
    return is_fact

def filter_factor_of_three(numbers):
    """Return all even numbers in list *numbers*."""
    return [n for n in numbers if factor_of_three(n)]
```

# Example: Moving on to the next unit

```
factor_three.py ✕ *test_factor_three.py ✕  
  
from factor_three import factor_of_three  
from factor_three import filter_factor_of_three  
  
def test_factor_of_three():  
    assert factor_of_three(3) == True  
    assert factor_of_three(4) == False  
    assert factor_of_three(5) == False  
    assert factor_of_three(6) == True  
    assert factor_of_three(0) == False  
    assert factor_of_three(3.0) == True  
    assert factor_of_three(6.6) == False  
    assert factor_of_three(3.00000001) == False  
    assert factor_of_three(6*1000) == True  
    assert factor_of_three(-9) == False  
  
def test_filter_factor_of_three():  
    in1 = [1,2,3,4,5,6,7,8,9]  
    out1 = [3,6,9]  
    in2 = [-2,-1,33,99]  
    out2 = [33,99]  
    print(filter_factor_of_three(in2))  
    assert (filter_factor_of_three(in1)) == out1  
    assert (filter_factor_of_three(in2)) == out2
```

# Example: Moving on to the next unit

```
factor_three.py  *test_factor_three.py
from factor_three import factor_of_three
from factor_three import filter_factor_of_three

def test_factor_of_three():
    assert factor_of_three(3) == True
    assert factor_of_three(4) == False
    assert factor_of_three(5) == False
    assert factor_of_three(6) == True
    assert factor_of_three(0) == False
    assert factor_of_three(3.0) == True
    assert factor_of_three(6.6) == False
    assert factor_of_three(3.00000001) == False
    assert factor_of_three(6*1000) == True
    assert factor_of_three(-9) == False

def test_filter_factor_of_three():
    in1 = [1,2,3,4,5,6,7,8,9]
    out1 = [3,6,9]
    in2 = [-2,-1,33,99]
    out2 = [33,99]
    print(filter_factor_of_three(in2))
    assert (filter_factor_of_three(in1)) == out1
    assert (filter_factor_of_three(in2)) == out2
```

```
Terminal
File Edit View Search Terminal Help

[bash]~/Desktop/nose_unittest >nosetests -v
test_factor_three.test_factor_of_three ... ok
test_factor_three.test_filter_factor_of_three ... ok

-----
Ran 2 tests in 0.008s

OK
[bash]~/Desktop/nose_unittest >
```

# More Resources

- Titus' tutorial: An Extended Introduction to the nose Unit Testing Framework  
<http://ivory.idyll.org/articles/nose-intro.html>
- Software Carpentry Video Tutorial: Unit Testing (nose)  
<http://software-carpentry.org/v4/test/unit.html>
- Jeff Knupp's Tutorial (unittest)  
<http://www.jeffknupp.com/blog/2013/12/09/improve-your-python-understanding-unit-testing/>