TDD in Python

An introduction to Test-Driven Development

About me

- Stephen Semeniuk
- Software architect at Intelligent Imaging Systems
- Developing software since 1981 (yikes!)
- Working with Python since 2002

How does TDD work?

- Step 0: write just enough code to have a sense of what you're doing
- Step 1: write your unit test
- Step 2: watch it fail
- Step 3: write your code; when your test passes, you're done
- Step 4: refactor (optional)

Write my tests first? That can't be right!

- How can I write tests when I don't know what my code is going to do?
  - It forces you to think about it, which is a good thing.
  - It forces you to write testable code.
- Won't that double the amount of code I write?
  - Yes, and maybe more.
  - You'll be doing it while your head is still in the problem.

The requisite contrived example

- Let's write an average function
- Write just enough code so we can start testing

Now we can write some tests

stephen@snpc-42

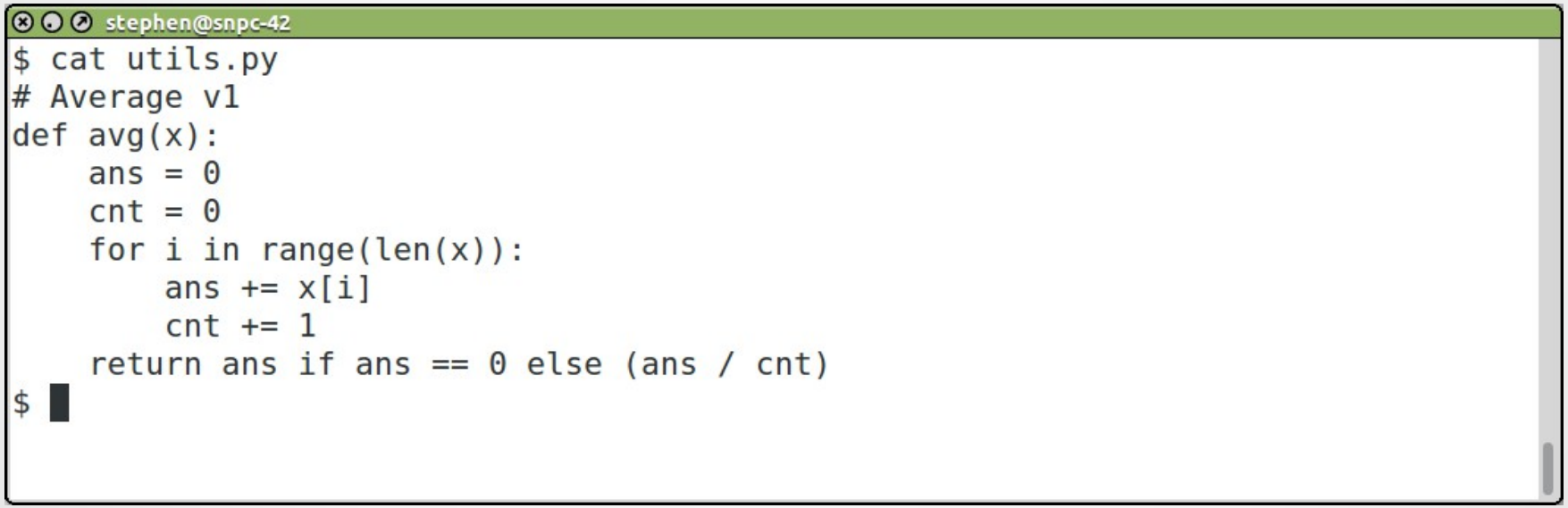
```
$ cat testavg.py
import unittest
from utils import avg

class AvgTest(unittest.TestCase):
    def test_int(self):
        '''Average of integers.'''
        self.assertEqual(avg([1, 2, 3]), 2)

    def test_dec(self):
        '''Average of decimals.'''
        self.assertEqual(avg([1.5, 2.5, 3.5]), 2.5)

    def test_neg(self):
        '''Average with negative values.'''
        self.assertEqual(avg([-1, 1, 0]), 0)

    def test_empty(self):
        '''Empty list should return 0.'''
        self.assertEqual(avg([]), 0)

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

Let's run the tests; we expect some failures

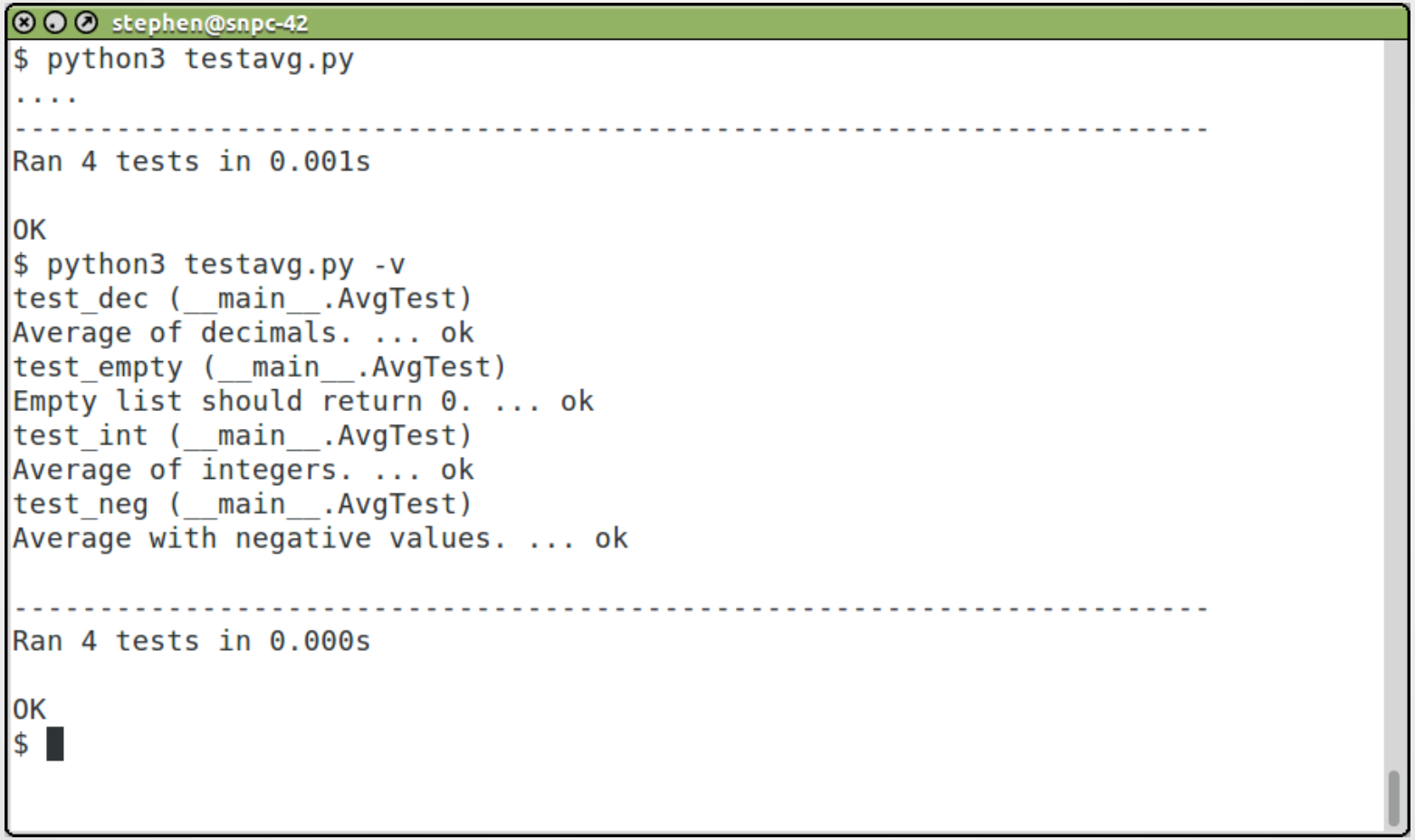
stephen@snpc-42

```
$ python3 testavg.py
F.F.
======================================================================
FAIL: test_dec (__main__.AvgTest)
Average of decimals.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "testavg.py", line 11, in test_dec
    self.assertEqual(avg([1.5, 2.5, 3.5]), 2.5)
AssertionError: 0 != 2.5

======================================================================
FAIL: test_int (__main__.AvgTest)
Average of integers.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "testavg.py", line 7, in test_int
    self.assertEqual(avg([1, 2, 3]), 2)
AssertionError: 0 != 2

----------------------------------------------------------------------
Ran 4 tests in 0.000s

FAILED (failures=2)
$
```

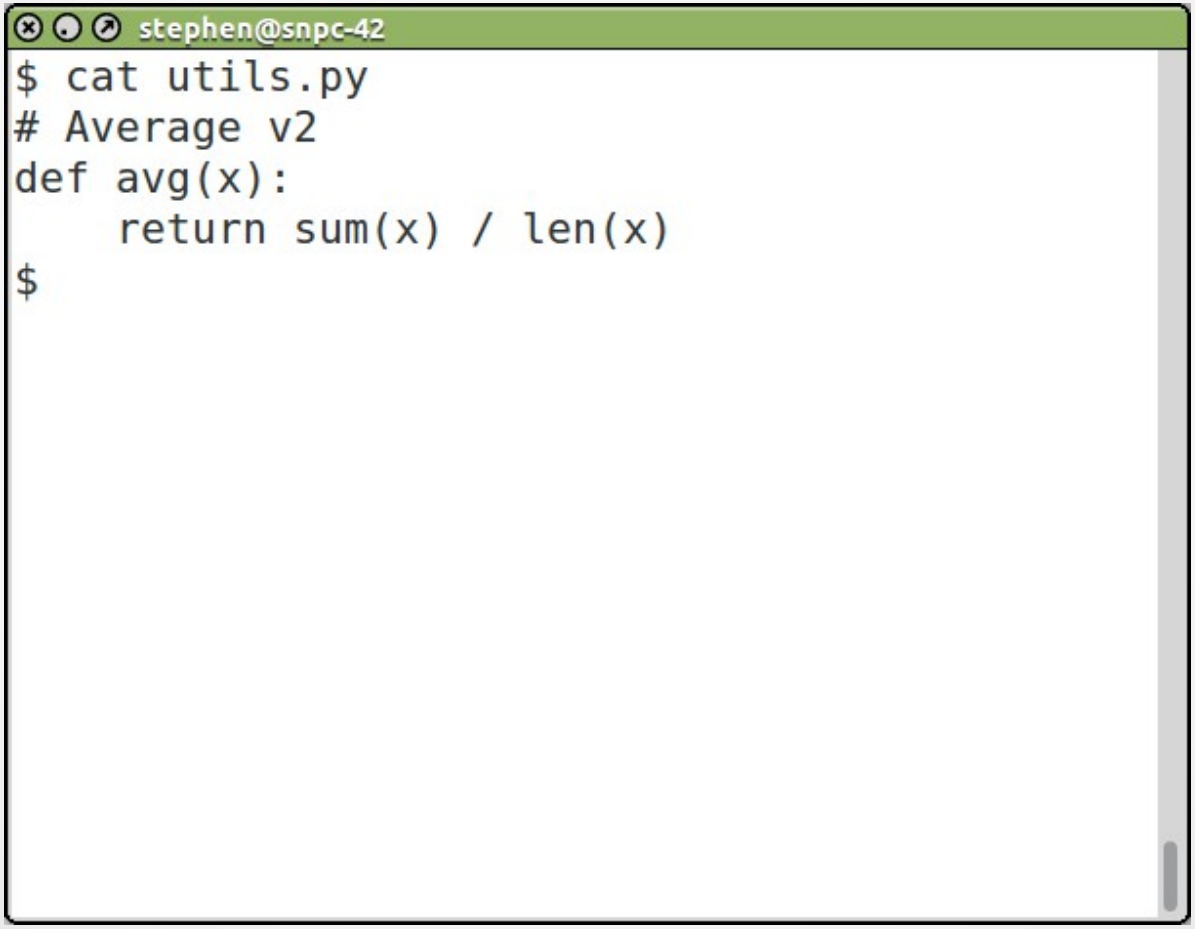
Now we can start coding!



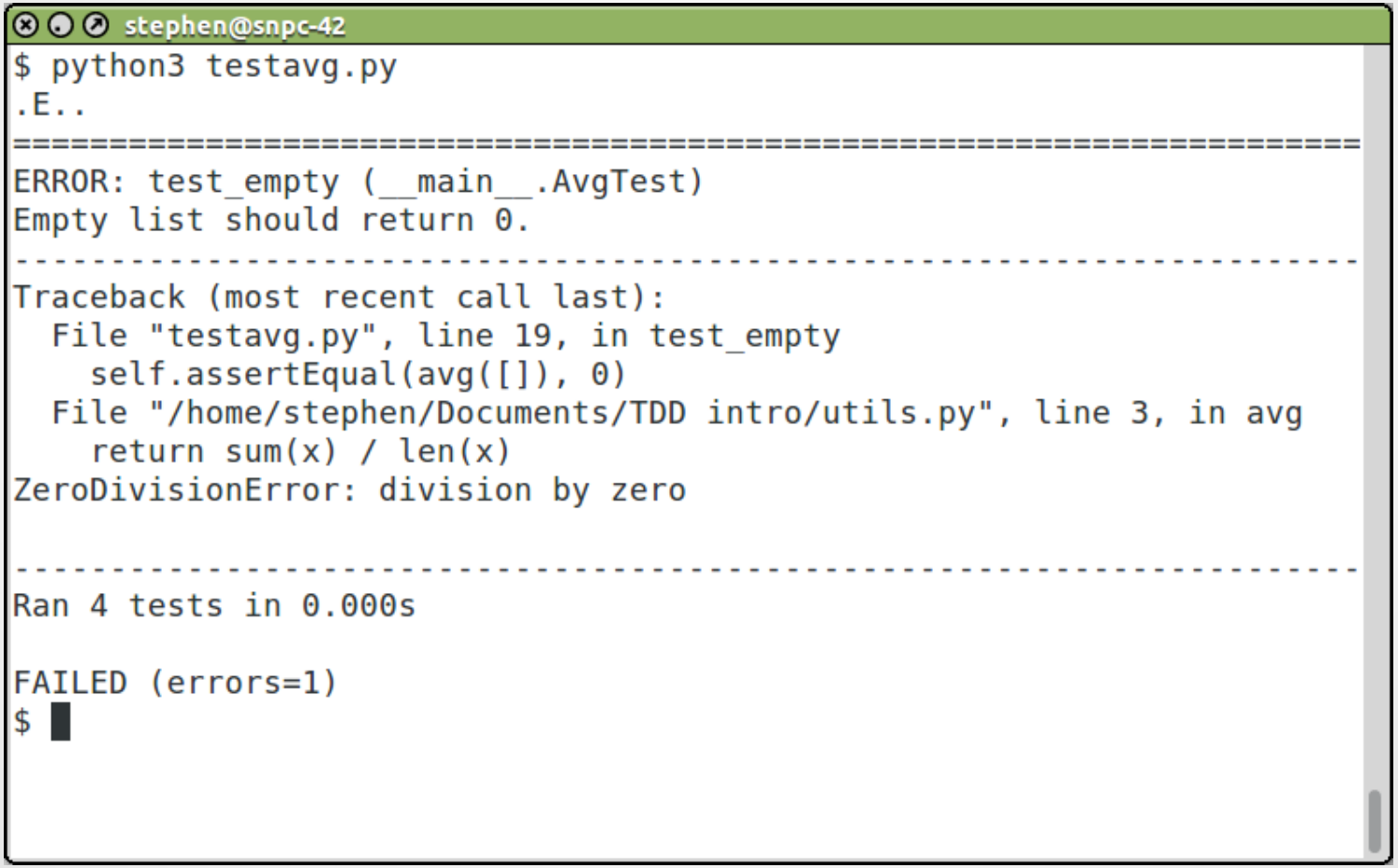
Let's see how we did



Refactor


stephen@snpc-42
$ cat utils.py
# Average v2
def avg(x):
    return sum(x) / len(x)
$

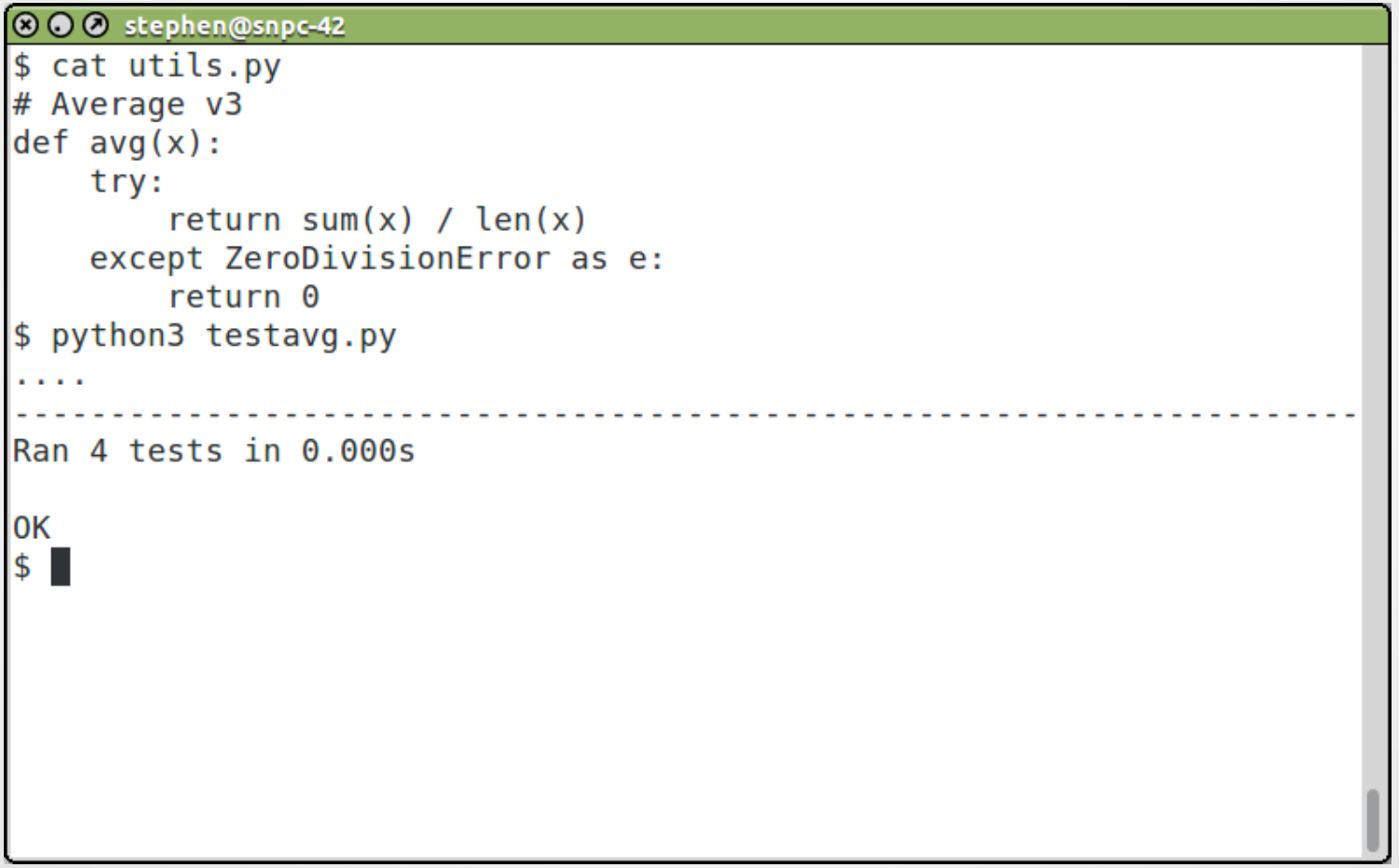
So, how did we do?



That's why we write unit tests

- So, what do we do now?
  - Fix the code?
  - Change the test?
- Don't do both!

Option A: Fix the code

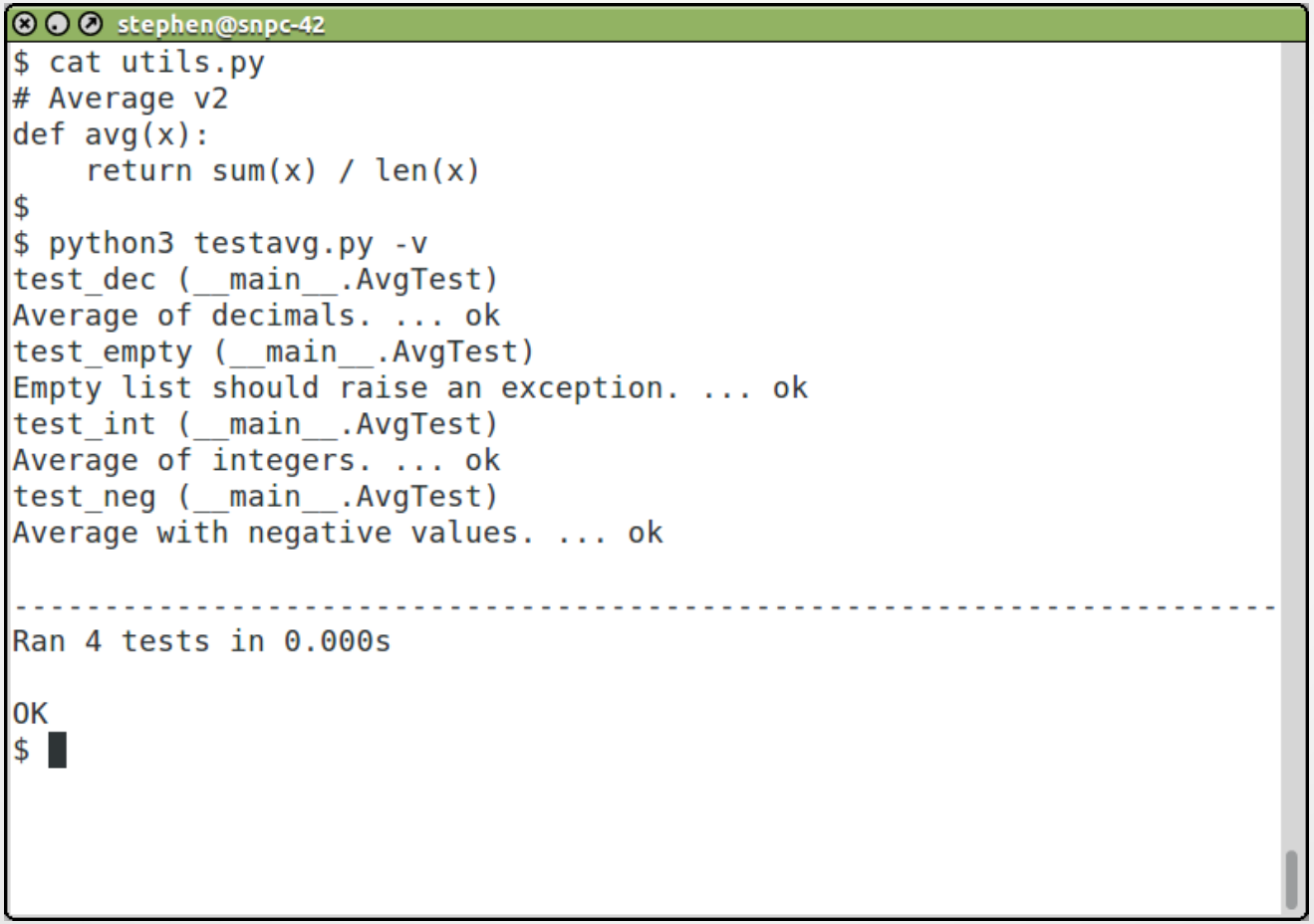


Option B: Fix the test

stephen@snpc-42

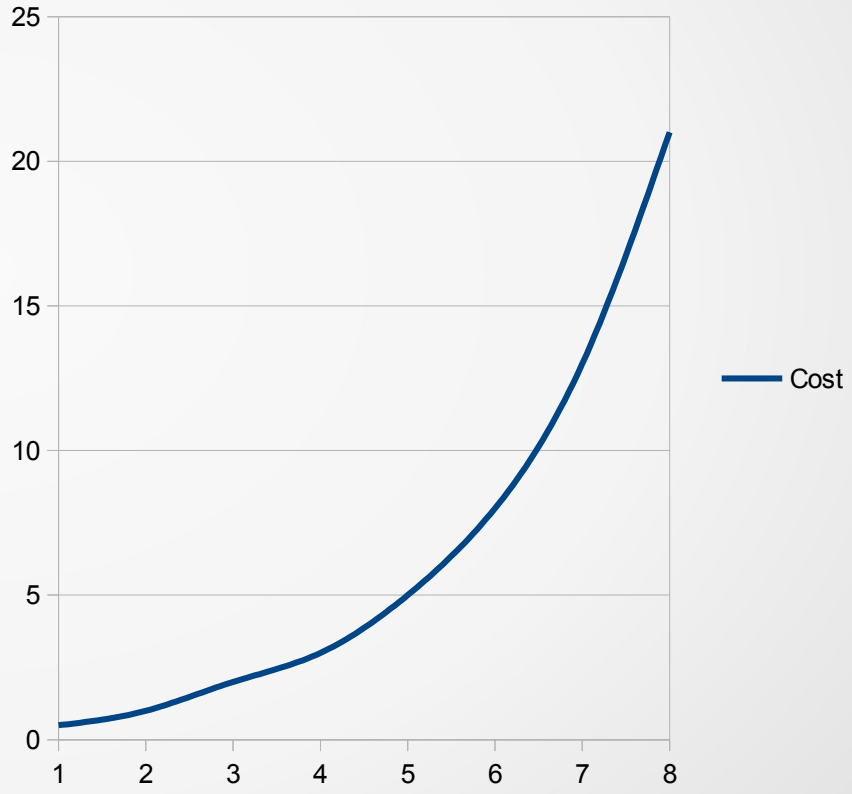
```
$ cat testavg.py
import unittest
from utils import avg

class AvgTest(unittest.TestCase):
    def test_int(self):
        '''Average of integers.'''
        self.assertEqual(avg([1, 2, 3]), 2)

    def test_dec(self):
        '''Average of decimals.'''
        self.assertEqual(avg([1.5, 2.5, 3.5]), 2.5)

    def test_neg(self):
        '''Average with negative values.'''
        self.assertEqual(avg([-1, 1, 0]), 0)

    def test_empty(self):
        '''Empty list should raise an exception.'''
        self.assertRaises(ZeroDivisionError, avg, [])

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

Option B continued...



Convince yourself

- The tests describe the behaviour of your code
- It's cheaper to test when you're writing the code than after the fact
- Multiple developers can work on a project
- Expect it to feel strange at first



Resources

- Test-driven development by example – Kent Beck
  - book
- pytest – Holger Krekel
  - Pycon presentation
- Google: test driven development python
- Google: unit testing ROI