

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

stephen@snp-42

```
$ cat utils.py
# Average v0
def avg(x):
    return 0
$ █
```

Now we can write some tests

```
stephen@snp-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@snp-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!

```
stephen@snp-42
$ cat utils.py
# Average v1
def avg(x):
    ans = 0
    cnt = 0
    for i in range(len(x)):
        ans += x[i]
        cnt += 1
    return ans if ans != 0 else (ans / cnt)
$ █
```


Let's see how we did

```
stephen@snp-42
$ python3 testavg.py
....
-----
Ran 4 tests in 0.001s

OK
$ python3 testavg.py -v
test_dec (__main__.AvgTest)
Average of decimals. ... ok
test_empty (__main__.AvgTest)
Empty list should return 0. ... ok
test_int (__main__.AvgTest)
Average of integers. ... ok
test_neg (__main__.AvgTest)
Average with negative values. ... ok

-----
Ran 4 tests in 0.000s

OK
$ █
```

Refactor

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

So, how did we do?

```
stephen@snp-42
$ python3 testavg.py
.E..
=====
ERROR: test_empty (__main__.AvgTest)
Empty list should return 0.
-----
Traceback (most recent call last):
  File "testavg.py", line 19, in test_empty
    self.assertEqual(avg([]), 0)
  File "/home/stephen/Documents/TDD intro/utils.py", line 3, in avg
    return sum(x) / len(x)
ZeroDivisionError: division by zero
-----

Ran 4 tests in 0.000s

FAILED (errors=1)
$ █
```

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

```
stephen@snp-42
$ cat utils.py
# Average v3
def avg(x):
    try:
        return sum(x) / len(x)
    except ZeroDivisionError as e:
        return 0
$ python3 testavg.py
....
-----
Ran 4 tests in 0.000s

OK
$ █
```

Option B: Fix the test

```
stephen@snp-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...

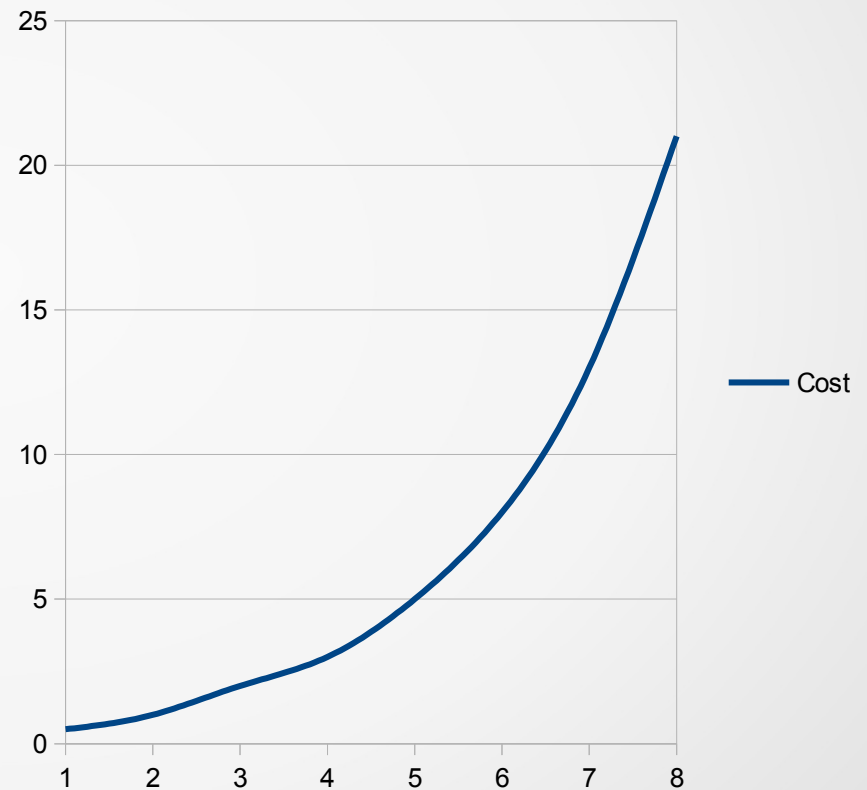
```
stephen@snp-42
$ cat utils.py
# Average v2
def avg(x):
    return sum(x) / len(x)
$
$ python3 testavg.py -v
test_dec (__main__.AvgTest)
Average of decimals. ... ok
test_empty (__main__.AvgTest)
Empty list should raise an exception. ... ok
test_int (__main__.AvgTest)
Average of integers. ... ok
test_neg (__main__.AvgTest)
Average with negative values. ... ok

-----
Ran 4 tests in 0.000s

OK
$ █
```

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