

# Unit Testing in Django

---



**Jamie Counsell**

SOFTWARE DEVELOPER

@jamiecounsell [www.jamiecounsell.me](http://www.jamiecounsell.me)



# Unit Testing



## Django unit testing options

- Unit testing best practices
- Writing our first tests
- `unittest.TestCase`
- `django.test.SimpleTestCase`

# Unit Tests Should Be

## Concise


Short and to the point, with a clear purpose – but never sacrificing usefulness

## Complete

Cover all edge cases, test only the features we care about, and have decent line coverage




# Test Discovery

▼  tasks

 tests.py

▼  tasks

▼  tests

 \_\_init\_\_.py

 test\_views.py

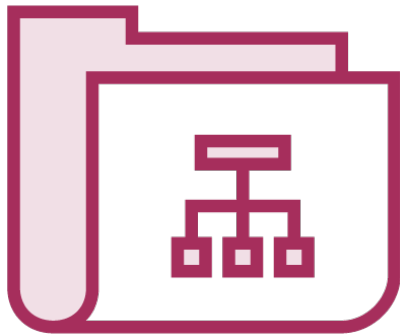
◀ Originally, we just had tests.py

◀ Now, Django will search all files under our app that match the pattern:

`test*.py`

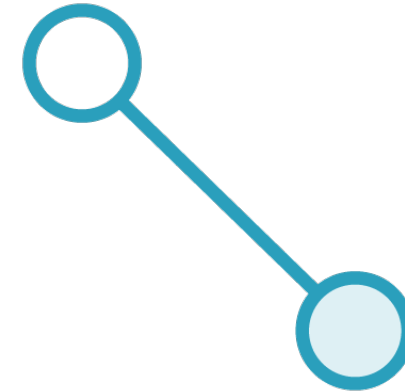


# Writing Many Tests



## Create Common Structure

Only the file pattern convention matters - the rest is up to us.  
Pick a system and stick with it



## Group Similar Tests

Spreading tests across files is great,  
but related tests should be in the  
same file – even as they grow

# unittest.TestCase

```
from unittest import TestCase
```

```
def add_x(base, x):  
    return base + x
```

```
class AdditionTest(TestCase):  
    def test_add_x_adds_x(self):  
        base = 10  
        x = 5  
        expected = 15  
        result = add_x(base, x)  
        self.assertTrue(  
            result == expected)
```

◀ Our method to test

◀ Test class has one goal, many tests

◀ Individual test methods test one case at a time



# django.test.SimpleTestCase

```
from django.test import SimpleTestCase
```

```
def div_x(base, x):  
    return base / x
```

```
class DivisionTest(SimpleTestCase):  
    def test_zero_division(self):  
        base = 20  
        x = 0  
  
        self.assertRaises(  
            ZeroDivisionError,  
            div_x, base = base, x = x)
```

- ◀ New method to test could cause some problems
- ◀ Create tests to cover interesting cases
- ◀ Extra assert methods provided by SimpleTestCase



# TestCase vs. SimpleTestCase

## `unittest.TestCase`

Built into Python's standard library

No dependencies

Great for testing pure python code

Faster

Limited features

## `django.test.SimpleTestCase`

Extends `unittest.TestCase`

Built into Django

Great for more advanced testing topics

Slower

Test template usage

Test assertions raised

Test HTTP response and/or redirect

Compare XML, HTML, and JSON





# Running Tests

```
$ ./manage.py test
```

```
$ ./manage.py test appname
```

```
$ ./manage.py test appname.tests
```

```
$ ./manage.py test appname.tests.test_helpers
```

```
$ ./manage.py test appname.tests.test_helpers.Atest
```

```
$ ./manage.py test appname.tests.test_helpers.Atest.test_a
```

◀ All tests (all apps)

◀ One Django app

◀ One test package

◀ One test module

◀ One test class

◀ One test



# Demo



## Writing unit tests

- Identify some unit test candidates
- Create our test file structure
- Write some unit tests using `unittest.TestCase`



# Django Test Classes

unittest

django.test

TestCase



SimpleTestCase



TransactionTestCase



LiveServerTestCase



TestCase



uni

integratio



# Summary



## Unit testing in Django

- Test discovery and file structure
- Unit tests with Python standard library
- Unit tests with Django's SimpleTestCase
- Running tests
- Test class options

