SubTests are the Best

PyOhio
July 29, 2017

Who am I?

- Dmitriy Chukhin
- Caktus Group
- Backend developer

caktus GROUP

Testing is important

- Make sure that code works
- Reduces technical debt
- Efficiency

caktus GROUP

We all know we should write tests, but...

Good tests are:

- readable
- thorough
- DRY (Don't Repeat Yourself)

Readable Tests

```
def test_following(self):
    """Following Profiles is tracked correctly."""
    profile1 = ProfileFactory()
    profile2 = ProfileFactory()
    profile3 = ProfileFactory()
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 0)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 0)
    self.assertEqual(profile3.followed_by.count(), 0)
```

caktus GROUP

```
def test_following(self):
    """Following Profiles is tracked correctly."""
    profile1 = ProfileFactory()
    profile2 = ProfileFactory()
    profile3 = ProfileFactory()
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 0)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 0)
    self.assertEqual(profile3.followed_by.count(), 0)
    profile1.following.add(profile2)
    profile3.following.add(profile1)
    self.assertEqual(profile1.following.count(), 1)
    self.assertEqual(profile1.following.first(), profile2)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 1)
    self.assertEqual(profile2.followed_by.first(), profile1)
    self.assertEqual(profile3.following.count(), 1)
    self.assertEqual(profile3.following.first(), profile1)
    self.assertEqual(profile3.followed_by.count(), 0)
```

caktus GROUP

```
def test_following(self):
    """Following Profiles is tracked correctly."""
    profile1 = ProfileFactory()
    profile2 = ProfileFactory()
    profile3 = ProfileFactory()
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 0)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 0)
    self.assertEqual(profile3.followed_by.count(), 0)
    profile1.following.add(profile2)
    profile3.following.add(profile1)
    self.assertEqual(profile1.following.count(), 1)
    self.assertEqual(profile1.following.first(), profile2)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 1)
    self.assertEqual(profile2.followed_by.first(), profile1)
    self.assertEqual(profile3.following.count(), 1)
    self.assertEqual(profile3.following.first(), profile1)
    self.assertEqual(profile3.followed_by.count(), 0)
    profile1.following.remove(profile2)
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 1)
    self.assertEqual(profile3.following.first(), profile1)
    self.assertEqual(profile3.followed_by.count(), 0)
```

caktus GROUP

```
def test_following(self):
    """Following Profiles is tracked correctly."""
    profile1 = ProfileFactory()
    profile2 = ProfileFactory()
    profile3 = ProfileFactory()

    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 0)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 0)
    self.assertEqual(profile3.followed_by.count(), 0)

    profile1.following.add(profile2)
    profile3.following.add(profile1)
    self.assertEqual(profile1.following.count(), 1)
    self.assertEqual(profile1.following.first(), profile2)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 1)
    self.assertEqual(profile2.followed_by.first(), profile1)
    self.assertEqual(profile3.following.count(), 1)
    self.assertEqual(profile3.following.first(), profile1)
    self.assertEqual(profile3.followed_by.count(), 0)

    profile1.following.remove(profile2)
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 1)
    self.assertEqual(profile3.following.first(), profile1)
    self.assertEqual(profile3.followed_by.count(), 0)
```

1

2

3

4

caktus GROUP

```
def test_following(self):
    """Following Profiles is tracked correctly."""
    # Create 3 Profiles
    profile1 = ProfileFactory()
    profile2 = ProfileFactory()
    profile3 = ProfileFactory()

    # No followers
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 0)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
    self.assertEqual(profile3.following.count(), 0)
    self.assertEqual(profile3.followed_by.count(), 0)

    # Several Profiles follow other Profiles
    profile1.following.add(profile2)
    profile3.following.add(profile1)
    self.assertEqual(profile1.following.count(), 1)
    self.assertEqual(profile1.following.first(), profile2)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 1)
    self.assertEqual(profile2.followed_by.first(), profile1)
    self.assertEqual(profile3.following.count(), 1)
    self.assertEqual(profile3.following.first(), profile1)
    self.assertEqual(profile3.followed_by.count(), 0)

    # Removing followers
    profile1.following.remove(profile2)
    self.assertEqual(profile1.following.count(), 0)
    self.assertEqual(profile1.followed_by.count(), 1)
    self.assertEqual(profile1.followed_by.first(), profile3)
    self.assertEqual(profile2.following.count(), 0)
    self.assertEqual(profile2.followed_by.count(), 0)
```

caktus GROUP

```
def test_following(self):
    """Following Profiles is tracked correctly."""
    # Create 3 Profiles
    profile1 = ProfileFactory()
    profile2 = ProfileFactory()
    profile3 = ProfileFactory()

    with self.subTest('No followers'):
        self.assertEqual(profile1.following.count(), 0)
        self.assertEqual(profile1.followed_by.count(), 0)
        self.assertEqual(profile2.following.count(), 0)
        self.assertEqual(profile2.followed_by.count(), 0)
        self.assertEqual(profile3.following.count(), 0)
        self.assertEqual(profile3.followed_by.count(), 0)

    with self.subTest('Several Profiles follow other Profiles'):
        # The profile1 follows profile2 and profile3 follows profile1
        profile1.following.add(profile2)
        profile3.following.add(profile1)

        # Now profile1 is following profile2
        self.assertEqual(profile1.following.count(), 1)
        self.assertEqual(profile1.following.first(), profile2)
        # Now profile1 is being followed by profile3
        self.assertEqual(profile1.followed_by.count(), 1)
        self.assertEqual(profile1.followed_by.first(), profile3)
        self.assertEqual(profile2.following.count(), 0)
        # Now profile2 is being followed by profile1
        self.assertEqual(profile2.followed_by.count(), 1)
        self.assertEqual(profile2.followed_by.first(), profile1)
        # Now profile3 is following profile1
        self.assertEqual(profile3.following.count(), 1)
        self.assertEqual(profile3.following.first(), profile1)
        self.assertEqual(profile3.followed_by.count(), 0)

    with self.subTest('Removing followers'):
        # The profile1 stops following profile2
        profile1.following.remove(profile2)
```

caktus GROUP

Readability is important

Why not break it up into small tests?

Thorough tests

```
import unittest

from ourapp.functions import is_user_error


class IsUserErrorTestCase(unittest.TestCase):
    def test_yes(self):
        """User errors return True."""
        self.assertTrue(is_user_error(400))
        self.assertTrue(is_user_error(401))
        self.assertTrue(is_user_error(402))
        self.assertTrue(is_user_error(403))
        self.assertTrue(is_user_error(404))
        self.assertTrue(is_user_error(405))

    def test_no(self):
        """Non-user errors return False."""
        self.assertFalse(is_user_error(200))
        self.assertFalse(is_user_error(201))
        self.assertFalse(is_user_error(202))
        self.assertFalse(is_user_error(500))
        self.assertFalse(is_user_error(503))
```

caktus GROUP

```
def test_yes(self):
    """User errors return True."""
    self.assertTrue(is_user_error(400))
    self.assertTrue(is_user_error(401))
    self.assertTrue(is_user_error(402))
    self.assertTrue(is_user_error(403))
    self.assertTrue(is_user_error(404))
    self.assertTrue(is_user_error(405))
    self.assertTrue(is_user_error(406))
    self.assertTrue(is_user_error(407))
    self.assertTrue(is_user_error(408))
    self.assertTrue(is_user_error(409))
    self.assertTrue(is_user_error(410))

def test_no(self):
    """Non-user errors return False."""
    self.assertFalse(is_user_error(200))
    self.assertFalse(is_user_error(201))
    self.assertFalse(is_user_error(202))
    self.assertFalse(is_user_error(203))
    self.assertFalse(is_user_error(204))
    self.assertFalse(is_user_error(205))
    self.assertFalse(is_user_error(206))
    self.assertFalse(is_user_error(207))
    self.assertFalse(is_user_error(208))
    self.assertFalse(is_user_error(209))
    self.assertFalse(is_user_error(210))
```

caktus GROUP

```
def test_yes(self):
    """User errors return True."""
    for i in range(400, 500):
        self.assertTrue(is_user_error(i))

def test_no(self):
    """Non-user errors return False."""
    for i in range(200, 300):
        self.assertFalse(is_user_error(i))
    for i in range(500, 600):
        self.assertFalse(is_user_error(i))
```

caktus GROUP

```
======================================================================
FAIL: test_yes (IsUserErrorTestCase)
User errors return True.
----------------------------------------------------------------------
Traceback (most recent call last):
 File "test_example.py", line 10, in test_yes
   self.assertTrue(is_user_error(status_code))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

caktus GROUP

```
def test_yes(self):
    """User errors return True."""
    for status_code in range(400, 500):
        with self.subTest(status_code=status_code):
            self.assertTrue(is_user_error(status_code))

def test_no(self):
    """Non-user errors return False."""
    for status_code in range(200, 300):
        with self.subTest(status_code=status_code):
            self.assertFalse(is_user_error(status_code))

    for status_code in range(500, 600):
        with self.subTest(status_code=status_code):
            self.assertFalse(is_user_error(status_code))
```

caktus GROUP

```
======================================================================
FAIL: test_yes (IsUserErrorTestCase) (status_code=405)
User errors return True.
----------------------------------------------------------------------
Traceback (most recent call last):
 File "test_example.py", line 10, in test_yes
   self.assertTrue(is_user_error(status_code))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

caktus GROUP

```
======================================================================
FAIL: test_yes (IsUserErrorTestCase) (status_code=403)
User errors return True.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_example.py", line 20, in test_yes
    self.assertTrue(is_user_error(status_code))
AssertionError: False is not true

======================================================================
FAIL: test_yes (IsUserErrorTestCase) (status_code=405)
User errors return True.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_example.py", line 20, in test_yes
    self.assertTrue(is_user_error(status_code))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 2 tests in 0.009s

FAILED (failures=2)
```

```
======================================================================
FAIL: test_get_full_name (bluebird.tests.test_example.CustomUserTestCase)
Test the get_full_name() method.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/dchukhin/dev/6ft-forum/bluebird/tests/test_example.py", line 36, in test_get_full_name
    self.assertEqual(result, expected_value)
AssertionError: 'Jane ()' != 'Jane  ()'
- Jane ()
+ Jane  ()
?      +


----------------------------------------------------------------------
```

caktus GROUP

```
======================================================================
FAIL: test_get_full_name (bluebird.tests.test_example.CustomUserTestCase) (first_name='Jane', last_name='')
Test the get_full_name() method.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/dchukhin/dev/6ft-forum/bluebird/tests/test_example.py", line 19, in test_get_full_name
    self.assertEqual(result, expected_value)
AssertionError: 'Jane ()' != 'Jane  ()'
- Jane ()
+ Jane  ()
?      +


======================================================================
FAIL: test_get_full_name (bluebird.tests.test_example.CustomUserTestCase) (first name='', last name='Smith')
Test the get_full_name() method.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/dchukhin/dev/6ft-forum/bluebird/tests/test_example.py", line 19, in test_get_full_name
    self.assertEqual(result, expected_value)
AssertionError: '()' != ' Smith ()'
- ()
+  Smith ()
```

caktus GROUP

Thorough is important, and can be readable

DRY Tests

```
def test_people_endpoint_invalid_post_data(self):
    """Test POSTing to the people endpoint with invalid data."""
    response = self.client.post(
        self.url,
        data={"last_name": "Shmo", "address": "123 Fake Street"},
        content_type='application/json')
    self.assertEqual(response.status_code, 400)
    response = self.client.post(
        self.url,
        data={"first_name": "Joe", "address": "123 Fake Street"},
        content_type='application/json')
    self.assertEqual(response.status_code, 400)
    response = self.client.post(
        self.url,
        data={"first_name": "Joe", "last_name": "Shmo"},
        content_type='application/json')
    self.assertEqual(response.status_code, 400)
```

caktus GROUP

```
def test_people_endpoint_invalid_post_data(self):
    """Test POSTing to the people endpoint with invalid data."""
    # POSTing with the first_name missing
    response = self.client.post(
        self.url,
        data={"last_name": "Shmo", "address": "123 Fake Street"},
        content_type='application/json')
    self.assertEqual(response.status_code, 400)

    # POSTing with the last_name missing
    response = self.client.post(
        self.url,
        data={"first_name": "Joe", "address": "123 Fake Street"},
        content_type='application/json')
    self.assertEqual(response.status_code, 400)

    # POSTing with the address missing
    response = self.client.post(
        self.url,
        data={"first_name": "Joe", "last_name": "Shmo"},
        content_type='application/json')
    self.assertEqual(response.status_code, 400)
```

caktus GROUP

```
def get_minimum_required_data(self):
    """Return the minimum data required for a successful POST."""
    return {
        "first_name": "Joe",
        "last_name": "Shmo",
        "address": "123 Fake Street",
    }

def test_people_endpoint_invalid_post_data(self):
    """Test POSting to the people endpoint with invalid data."""
    missing_subtests = (
        # A tuple of (field_name, subtest_description)
        ('first_name', 'Missing the first_name field'),
        ('last_name', 'Missing the last_name field'),
        ('address', 'Missing the address field'),
    )
    for field_name, subtest_description in missing_subtests:
        with self.subTest(subtest_description):
            # Remove the missing field from the minimum_required_data
            data = self.get_minimum_required_data()
            data.pop(field_name)

            # POST with the missing field name
            response = self.client.post(self.url, data=data, content_type='application/json')
            self.assertEqual(response.status_code, 400)
```

caktus GROUP

DRY testing code is important

Lessons to learn from subTests

- Readability matters
- Thorough tests can be readable and/or short
- DRY tests are important

Do subTests solve all our testing problems?

Ways to write bad code using subTests

- comments
- variable names
- massive amounts of setup
- others

caktus GROUP

Other testing libraries?

Sure!

Contact

Dmitriy Chukhin

- github.com/dchukhin
- Twitter: @dchukhin1

Caktus Group

- caktusgroup.com
- github.com/caktus
- Twitter: @caktusgroup

caktus GROUP