Mocking Strategies

Daniel Davis

Please don’t mock me after this presentation…

Who Am I?

- Daniel Davis
- Software Developer for 8 years
- Fun Fact:
  - I ran the “Jingle All The Way” 5k dressed as a Giant Gingerbread man


7AM-7PM
1984
1978

Really though, who are you?

- Came from Java world
- Python developer for 2 years
- DevOps
  - Lots of work with automation and quality
- Passionate about quality!
  - Doesn’t happen overnight…

excella CONSULTING

Testing Journey

I want to be a better developer

- Became a Certified Scrum Developer (CSD) about 2 years ago
- Finally learned about craftsmanship and writing better tests
  - Basically black magic
- Learned in Java…

Finding Python

- Switching to Python was jarring
- Kept wondering about writing unit tests
- Dealing with complicated frameworks like Django

excella CONSULTING

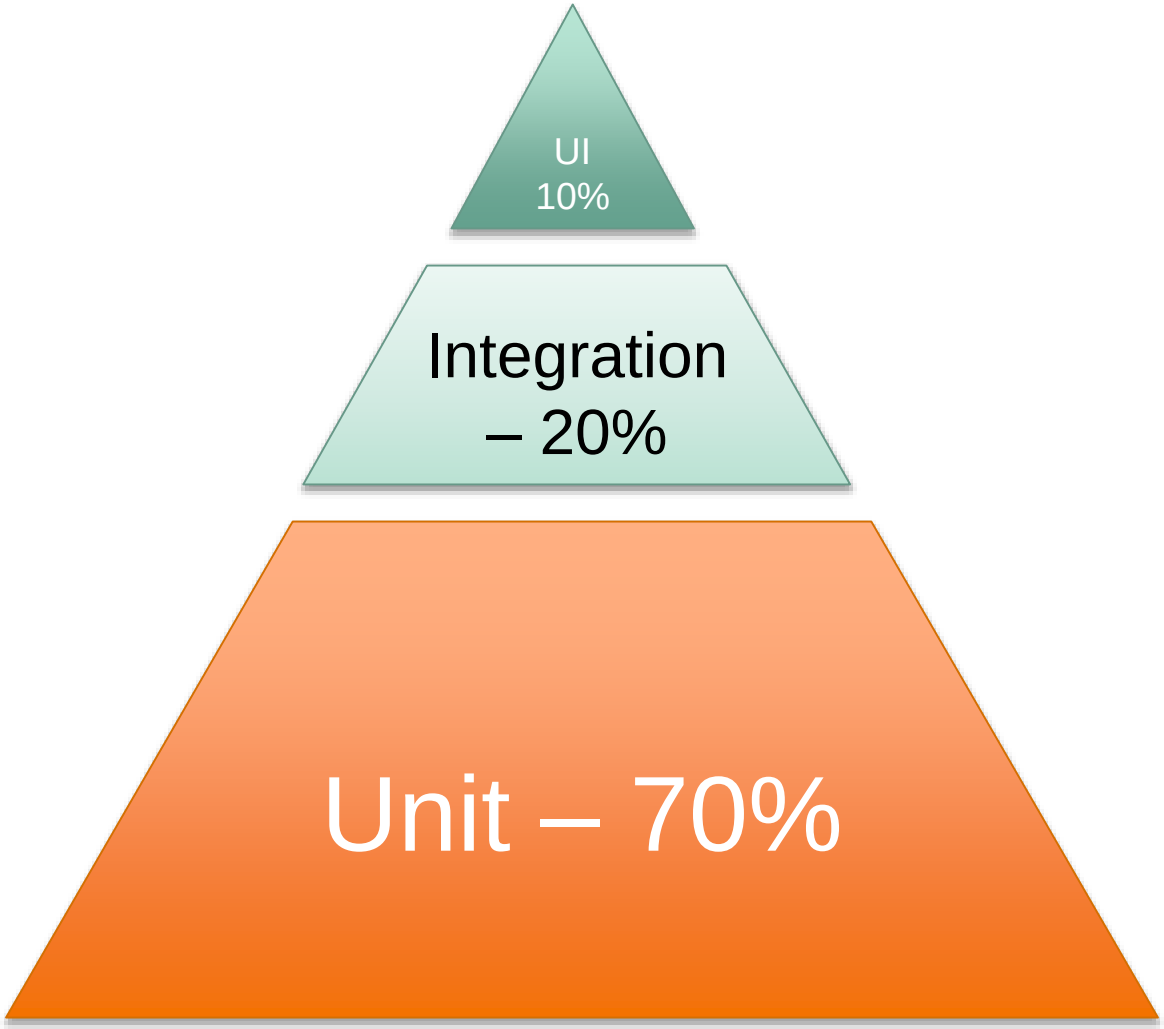
Unit testing? Mocks?

- Asked colleagues
  - Many had the same questions
- Decided to learn…
- Wanted to share with the community

excella CONSULTING

Let’s Talk About Unit Tests…

Testing Types

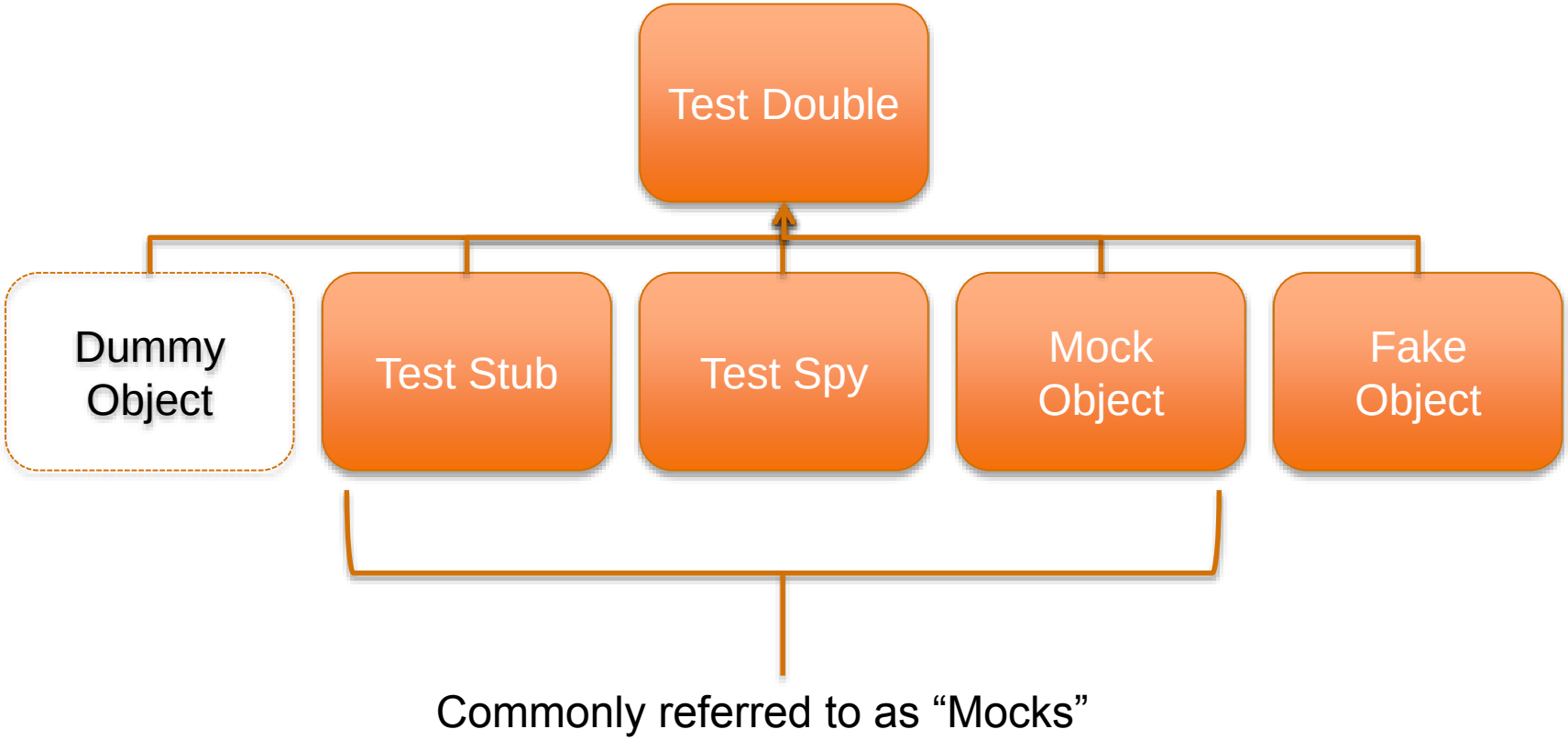


Great Unit Testing Myths

- “They’re good when the problem is easy”
  - A rabbit hole of testing
- “I spend too much time writing lots of code to test, so I give up”
- “There’s just some stuff you can’t unit test”

Mocking makes unit testing easier!

What Are Mocks?



What Are Mocks?

- Stubs
  - Provide a canned response to method calls
- Spy
  - Real objects that behave like normal except when a specific condition is met
- Mocks
  - Verifies behavior (calls) to a method

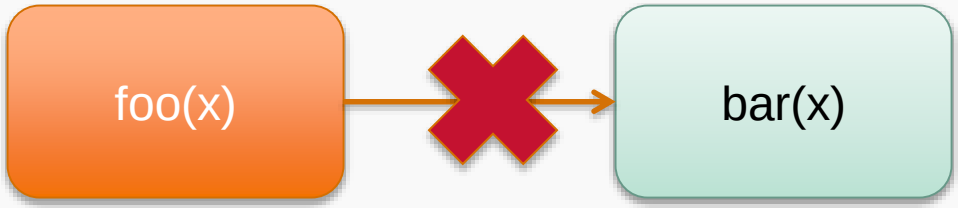
excella CONSULTING

Blah Blah Blah professor Dan…

Problems Mocks Solve

- Eliminates dependencies in the CUT (class under test)
  - Isolated Unit Tests

```
#!/usr/bin/env python

def foo(x):
    y = bar(x)
    if y > 10:
        return x+y
    retun x-y
```



Problems Mocks Solve

- Tests methods that have no return value

```
#!/usr/bin/env python

def foo(x):
    if x > 10:
        bar(x)
    else:
        something_else(x)
```

How do we know that bar(x) has been called?

excella CONSULTING

Problems Mocks Solve

- Tests error handling

```
#!/usr/bin/env python

def foo(filename):
    try:
        return parse_large_file(filename)
    except MemoryError:
        return ""
```

How do we generate this exception???

excella CONSULTING

Other Reasons Mocks Are Important

- Eliminate dependency on database calls
  - Speed up testing!
- Reduce test complexity
  - Don’t have to write complex logic to handle behavior of methods not under test
- Don’t have to wait to implement other methods

excella CONSULTING

Ok, I’m sold…
Show me how to actually do this…

What Are The Python Options?

- Mock (MagicMock)
  - Most robust, popular
  - Built-in as of Python 3.3!
- flexmock
  - Based on Ruby's flexmock
- mox
  - Similar to Java's EasyMock
- Mocker
- dingus
  - "record then assert" mocking library
- fudge
  - Similar to Mockito
- MiniMock
  - Simple mocking with DocTest

excella CONSULTING

Sample Problem

Valentine's Day Edition

Problem:
Tinder Competitor

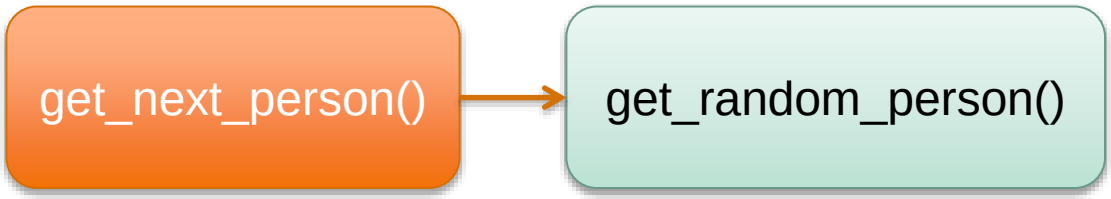
Get yo' container on…

Problem: “Docker” dating app

- Create a method to return a new, random ~~victim~~ candidate
  - Must not show the same person
  - Must not show someone the user has already “swiped” on

Easy enough…

```
def get_next_person(user):
    person = get_random_person()
    while person in user['people_seen']:
        person = get_random_person()
    return person
```



"Surely no one could have seen EVERYONE in the database!!!"
- The Intern

Write a Unit Test…

```
def test_new_person():
    # arrange
    user = {'people_seen': []}
    expected_person = 'Katie'

    # action
    actual_person = get_next_person(user)

    # assert
    assert_equals(actual_person, expected_person)
```

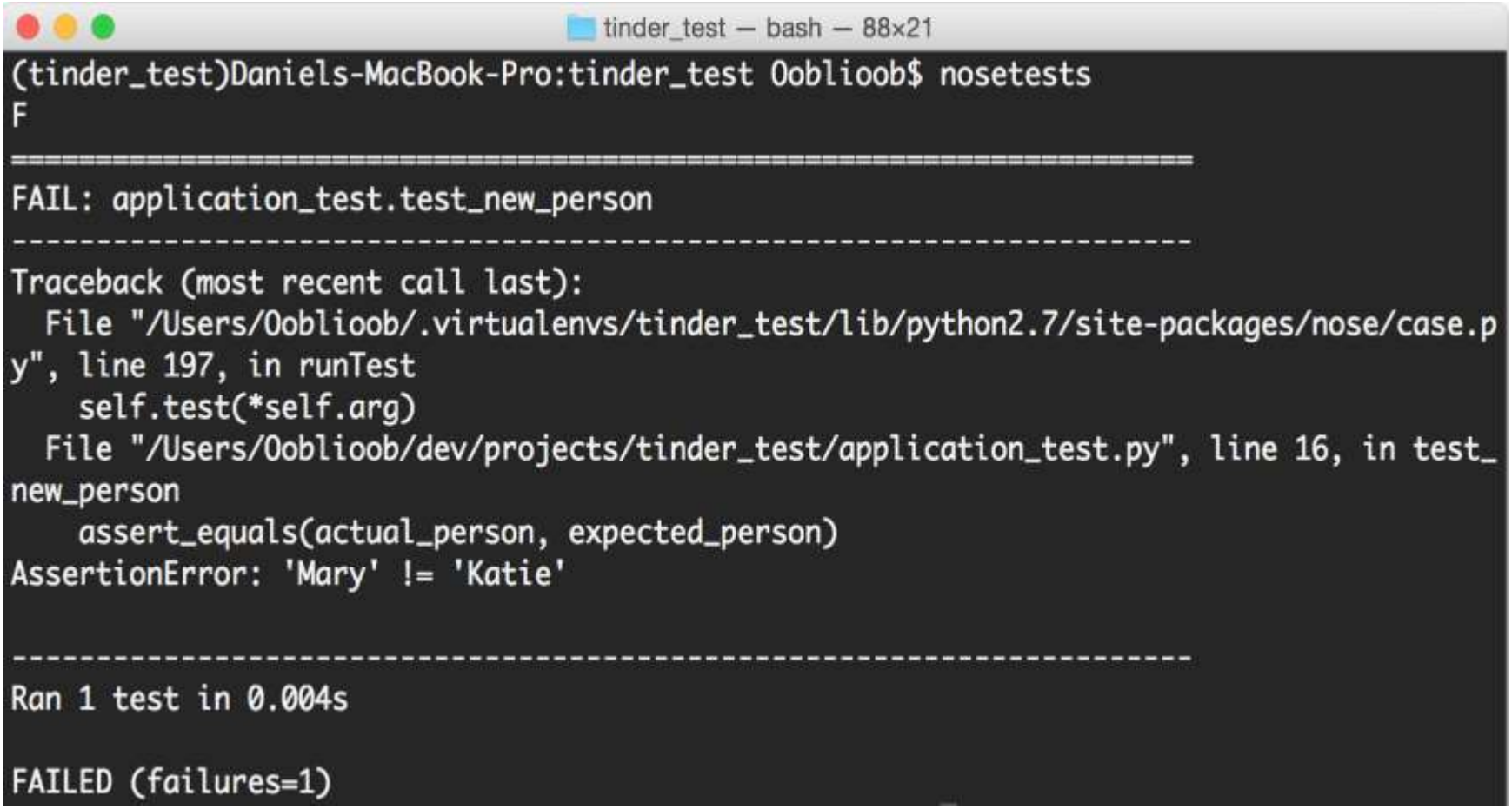

excella CONSULTING

It works!!!

tinder_test — bash — 88×21

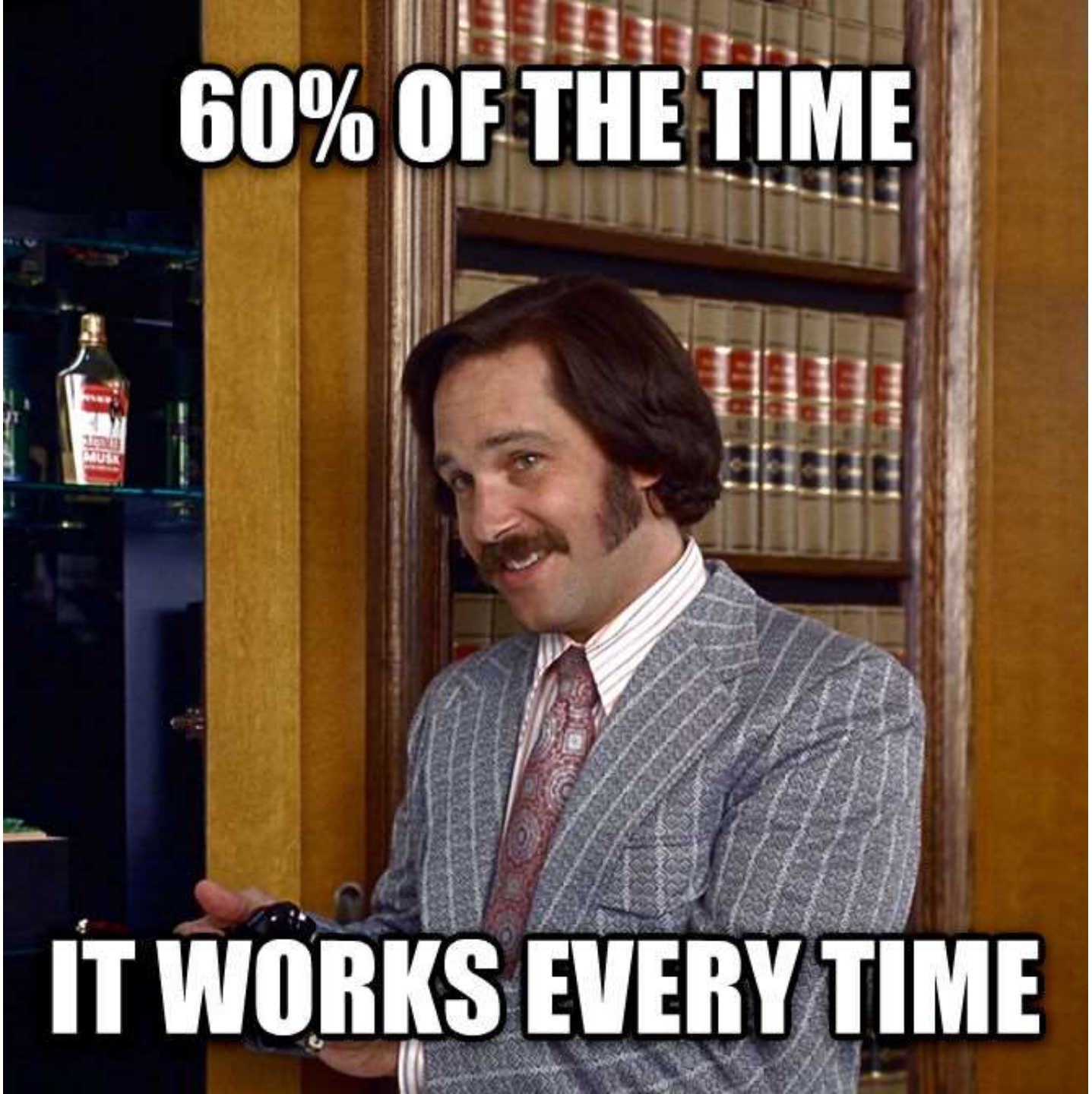
```
(tinder_test)Daniels-MacBook-Pro:tinder_test 0oblioob$ nosetests
.
----------------------------------------------------------------------
Ran 1 test in 0.005s

OK
```


excella CONSULTING


tinder_test — bash — 88×21
(tinder_test)Daniels-MacBook-Pro:tinder_test Ooblioob$ nosetests
F
==
FAIL: application_test.test_new_person

Traceback (most recent call last):
File "/Users/Ooblioob/.virtualenvs/tinder_test/lib/python2.7/site-packages/nose/case.py", line 197, in runTest
self.test(*self.arg)
File "/Users/Ooblioob/dev/projects/tinder_test/application_test.py", line 16, in test_new_person
assert_equals(actual_person, expected_person)
AssertionError: 'Mary' != 'Katie'

Ran 1 test in 0.004s
FAILED (failures=1)


60% OF THE TIME
IT WORKS EVERY TIME

Easy enough…

```
def get_next_person(user):
    person = get_random_person()
    while person in user['people_seen']:
        person = get_random_person()
    return person
```

What if knew the result of *get_random_person()*???

excella CONSULTING

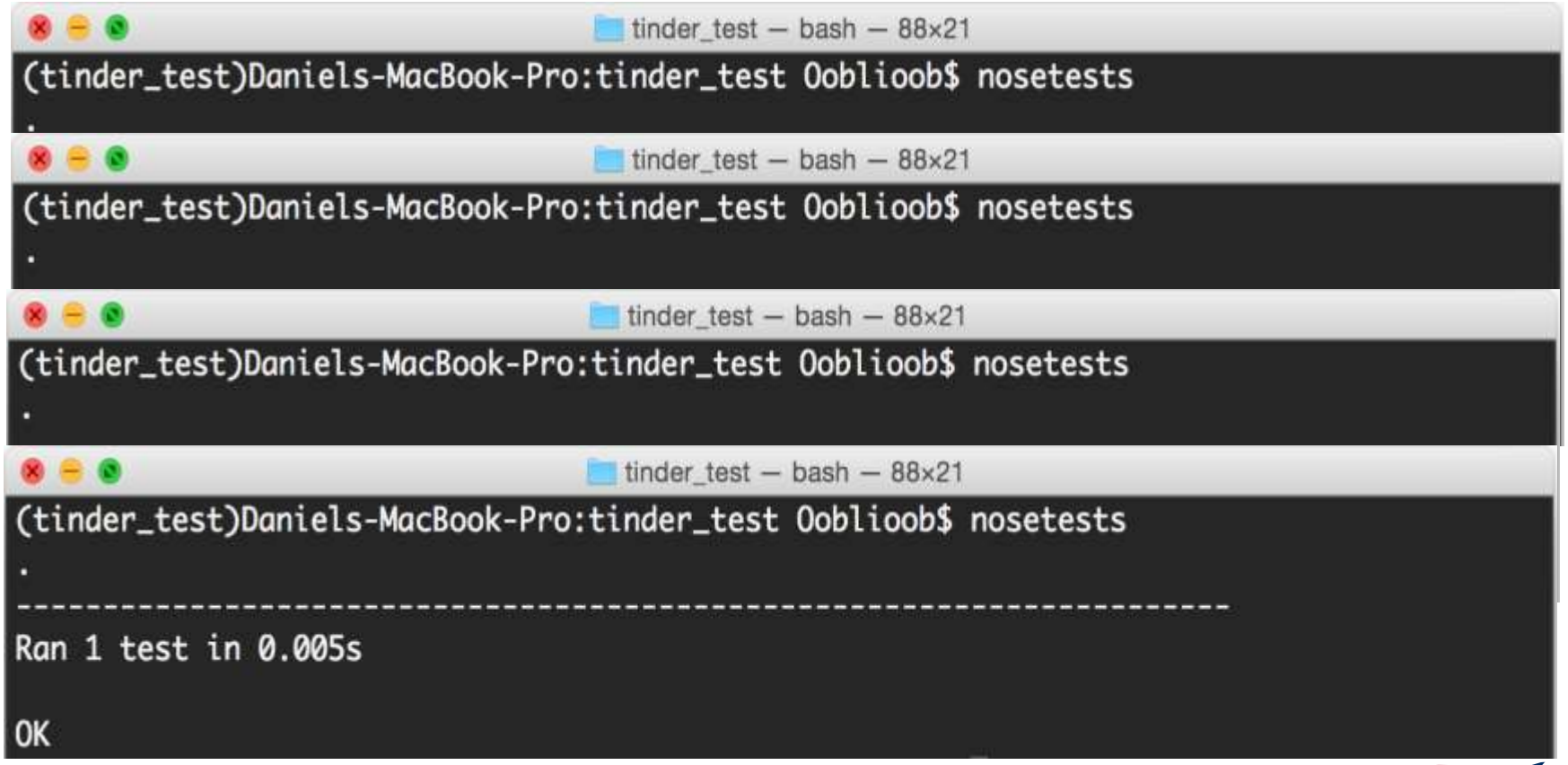

MOCK
ALL THE DEPENDENCIES!

Patching

Module.attribute

Mock method

```
from mock import patch

@patch("application.get_random_person")
def test_new_person(mock_get_rand_person):
    # arrange
    user = {'people_seen': []}
    expected_person = 'Katie'
    mock_get_rand_person.return_value = 'Katie'

    # action
    actual_person = get_next_person(user)

    # assert
    assert_equals(actual_person, expected_person)
```



It works EVERY SINGLE TIME!!!

tinder_test — bash — 88×21

```
(tinder_test)Daniels-MacBook-Pro:tinder_test Ooblioob$ nosetests
.
```

tinder_test — bash — 88×21

```
(tinder_test)Daniels-MacBook-Pro:tinder_test Ooblioob$ nosetests
.
```

tinder_test — bash — 88×21

```
(tinder_test)Daniels-MacBook-Pro:tinder_test Ooblioob$ nosetests
.
```

tinder_test — bash — 88×21

```
(tinder_test)Daniels-MacBook-Pro:tinder_test Ooblioob$ nosetests
.
----------------------------------------------------------------------
Ran 1 test in 0.005s

OK
```


excella CONSULTING

Variations on a theme

```
class Application:
    def get_next_person(self, user):
        person = self.get_random_person()
        while person in user['people_seen']:
            person = self.get_random_person()
        return person
```


excella CONSULTING

Variations on a theme

```
@patch.object(Application, "get_random_person")
def test_new_person(mock_get_rand_person):
    # arrange
    app = Application()
    user = {'people_seen': []}
    expected_person = 'Katie'
    mock_get_rand_person.return_value = 'Katie'

    # action
    actual_person = app.get_next_person(user)

    # assert
    assert_equals(actual_person, expected_person)
```

Variations on a theme

```
def test_new_person():
    # arrange
    app = Application()
    user = {'people_seen': []}
    expected_person = 'Katie'
    app.get_random_person = Mock() # or MagicMock()
    app.get_random_person.return_value = 'Katie'

    # action
    actual_person = app.get_next_person(user)

    # assert
    assert_equals(actual_person, expected_person)
```

Variations on a theme

```
def test_new_person():
    with patch.object(Application, "get_random_person") \
            as mock_get_random_person:
        # arrange
        app = Application()
        user = {'people_seen': []}
        expected_person = 'Katie'
        mock_get_random_person.return_value = 'Katie'

        # action
        actual_person = app.get_next_person(user)

        # assert
        assert_equals(actual_person, expected_person)
```

OMG, ContextManagers! WHAT???

excella CONSULTING

But what if we call it multiple times???

Different results on multiple calls

```
class Application:
    def get_next_person(self, user):
        person = self.get_random_person()
        while person in user['people_seen']:
            person = self.get_random_person()
        return person
```

- What if I want to test the while loop?

Uh…umm…

```
@patch.object(Application, "get_random_person")
def test_experienced_user(mock_get_rand_person):
    # arrange
    app = Application()
    user = {'people_seen': ['Sarah', 'Mary']}
    expected_person = 'Katie'
    mock_get_rand_person.return_value = ???

    # action
    actual_person = app.get_next_person(user)

    # assert
    assert_equals(actual_person, expected_person)
```

UMM…

Use *side_effect*

```
@patch.object(Application, "get_random_person")
def test_experienced_user(mock_get_rand_person):
    # arrange
    app = Application()
    user = {'people_seen': ['Sarah', 'Mary']}
    expected_person = 'Katie'
    mock_get_rand_person.side_effect = ['Mary', 'Sarah', 'Katie']

    # action
    actual_person = app.get_next_person(user)

    # assert
    assert_equals(actual_person, expected_person)
```


excella CONSULTING

Recap: What did we learn?

- Use patching / mocks to bring certainty to method calls
- Eliminates dependencies on other code
  - Even unfinished code!!!
- Lots of ways to do it, pick your favorite

excella CONSULTING

Mocking to Verify Behavior

Problem:
Matching in “Docker”

Problem: “Docker” matches

- When a user swipes right…
- If the other user “likes” them:
  - Send them both a message with contact info
- If the other user “dislikes” them:
  - Let the user down gently…
- If the other user hasn’t evaluated yet:
  - Display the “give it time” message

Implementation

```
def evaluate(person1, person2):
    if person1 in person2['likes']:
        send_email(person1)
        send_email(person2)
    elif person1 in person2['dislikes']:
        let_down_gently(person1)
    elif person1 not in person2['likes'] \
        and person1 not in person2['dislikes']:
        give_it_time(person1)
```

How do we test this??? No return values!!!

Behavior Verification

```
@patch("application.let_down_gently")
def test_person2_dislikes_person1(mock_let_down):
    # arrange
    person1 = 'Bill'
    person2 = {
        'likes': ['Sam', 'Joey'],
        'dislikes': ['Bill']
    }

    # action
    evaluate(person1, person2)

    # assert
    assert_equals(mock_let_down.call_count, 1)
```

What about checking parameters???

Verifying Parameters

```
@patch("application.let_down_gently")
def test_person2_dislikes_person1(mock_let_down):
    # arrange
    person1 = 'Bill'
    person2 = {
        'likes': ['Sam', 'Joey'],
        'dislikes': ['Bill']
    }

    # action
    evaluate(person1, person2)

    # assert
    mock_let_down.assert_called_once_with(person1)
```

Shouldn't we check the other methods too?

```
def evaluate(person1, person2):
    if person1 in person2['likes']:
        send_email(person1)
        send_email(person2)
    elif person1 in person2['dislikes']:
        let_down_gently(person1)
    elif person1 not in person2['likes'] \
        and person1 not in person2['dislikes']:
        give_it_time(person1)
```

We'd need to have multiple mocks to do that!!!

Multiple Mocks

```
@patch("application.send_email")
@patch("application.let_down_gently")
@patch("application.give_it_time")
def test_person2_dislikes_person1(mock_give_it_time,
                                  mock_let_down,
                                  mock_send_email):
    # arrange
    person1 = 'Bill'
    person2 = {'likes': ['Sam'], 'dislikes': ['Bill'] }

    # action
    evaluate(person1, person2)

    # assert
    mock_let_down.assert_called_once_with(person1)
    assert_equals(mock_give_it_time.call_count, 0)
    assert_equals(mock_send_email.call_count, 0)
```

Patch Multiple

```
@patch.multiple("application",
    send_email=DEFAULT,
    let_down_gently=DEFAULT,
    give_it_time=DEFAULT)
def test_person2_dislikes_person1_multi(send_email,
                                        let_down_gently,
                                        give_it_time):
    # arrange
    person1 = 'Bill'
    person2 = {'likes': ['Sam'], 'dislikes': ['Bill'] }

    # action
    evaluate(person1, person2)

    # assert
    let_down_gently.assert_called_once_with(person1)
    assert_equals(give_it_time.call_count, 0)
    assert_equals(send_email.call_count, 0)
```

Testing Multiple Calls…

```
def evaluate(person1, person2):
    if person1 in person2['likes']:
        send_email(person1)
        send_email(person2)
    elif person1 in person2['dislikes']:
        let_down_gently(person1)
    elif person1 not in person2['likes'] \
        and person1 not in person2['dislikes']:
        give_it_time(person1)
```

Multiple calls

```
@patch("application.send_email")
@patch("application.let_down_gently")
@patch("application.give_it_time")
def test_person2_likes_person1(mock_give_it_time,
                               mock_let_down,
                               mock_send_email):
    # arrange
    person1 = 'Bill'
    person2 = {'likes': ['Bill'], 'dislikes': ['Sam'] }

    # action
    evaluate(person1, person2)

    # assert
    first_call = mock_send_email.call_args_list[0]
    second_call = mock_send_email.call_args_list[1]
    assert_equals(first_call, call(person1))
    assert_equals(second_call, call(person2))
```

Whew!!!
Almost Done!

There will be beer soon!

Sweet, delicious beer!

Mocking built-ins and Exceptions

Simple JSON reader

```
def get_json(filename):
    try:
        return json.loads(open(filename).read())
    except (IOError,ValueError):
        return ""
```

How do we test something like this???

excella CONSULTING

Testing JSON reader

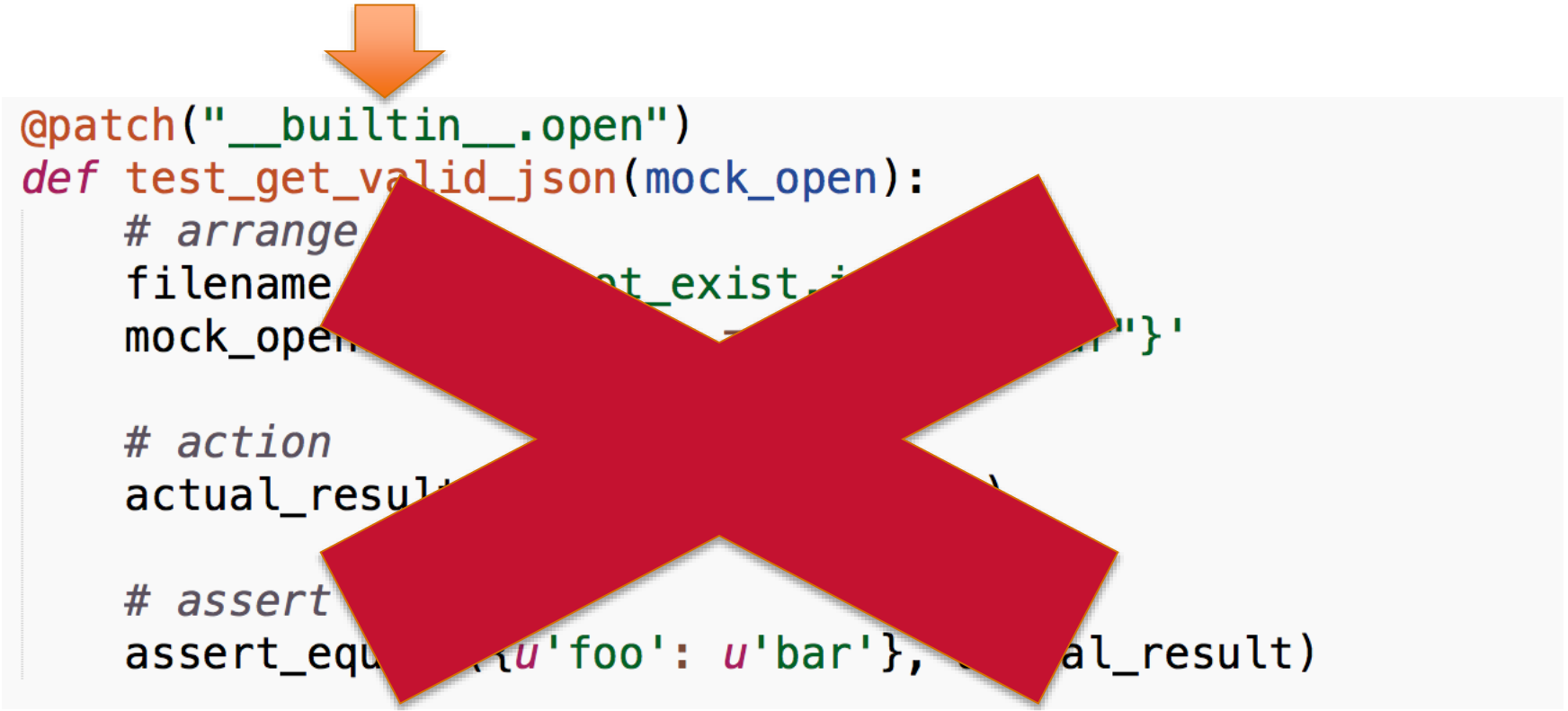
- Test parsing a valid file
- Test an IOError (i.e. file missing)
- Test a ValueError (i.e invalid json)

excella CONSULTING

How do I test *open()*

- Let's just create a sample file!

Can you even mock a builtin?



excella CONSULTING

What? Why Not???

- open() returns a File object
- open(filename).**read()**
- So we really need to mock File.read()
  - But it's an instance!!! Oh no!

Have you tried solving it with Mocks???

excella CONSULTING

Mocks returning Mocks? WAT???

```
@patch("__builtin__.open")
def test_get_valid_json(mock_open):
    # arrange
    filename = "does_not_exist.json"
    mock_file = Mock()
    mock_file.read.return_value = '{"foo": "bar"}'
    mock_open.return_value = mock_file

    # action
    actual_result = get_json(filename)

    # assert
    assert_equals({u'foo': u'bar'}, actual_result)
```


excella CONSULTING

What about error handling?

What about error handling?

```
@patch("__builtin__.open")
def test_get_json_ioerror(mock_open):
    # arrange
    filename = "does_not_exist.json"
    mock_open.side_effect = IOError

    # action
    actual_result = get_json(filename)

    # assert
    assert_equals('', actual_result)
```


excella CONSULTING

What about ValueError?

```
@patch("__builtin__.open")
@patch("json.loads")
def test_get_json_ValueError(mock_loads, mock_open):
    # arrange
    filename = "does_not_exist.json"
    mock_file = Mock()
    mock_file.read.return_value = '{"foo": "bar"}'
    mock_open.return_value = mock_file
    mock_loads.side_effect = ValueError

    # action
    actual_result = get_json(filename)

    # assert
    assert_equals('', actual_result)
```

Wrap Up:
Key Take-Aways

Remember This!

- Mocking makes writing unit tests simpler
  - Eliminates dependencies
  - Verifies behavior
  - Tests error handling

- You just need some practice!

Try It On Your Own

- http://mock.readthedocs.org/en/latest/
- Pip Install Mock
- Create a simple class, then write tests!

excella CONSULTING

Let’s Go Write Some Tests!!!

Questions?