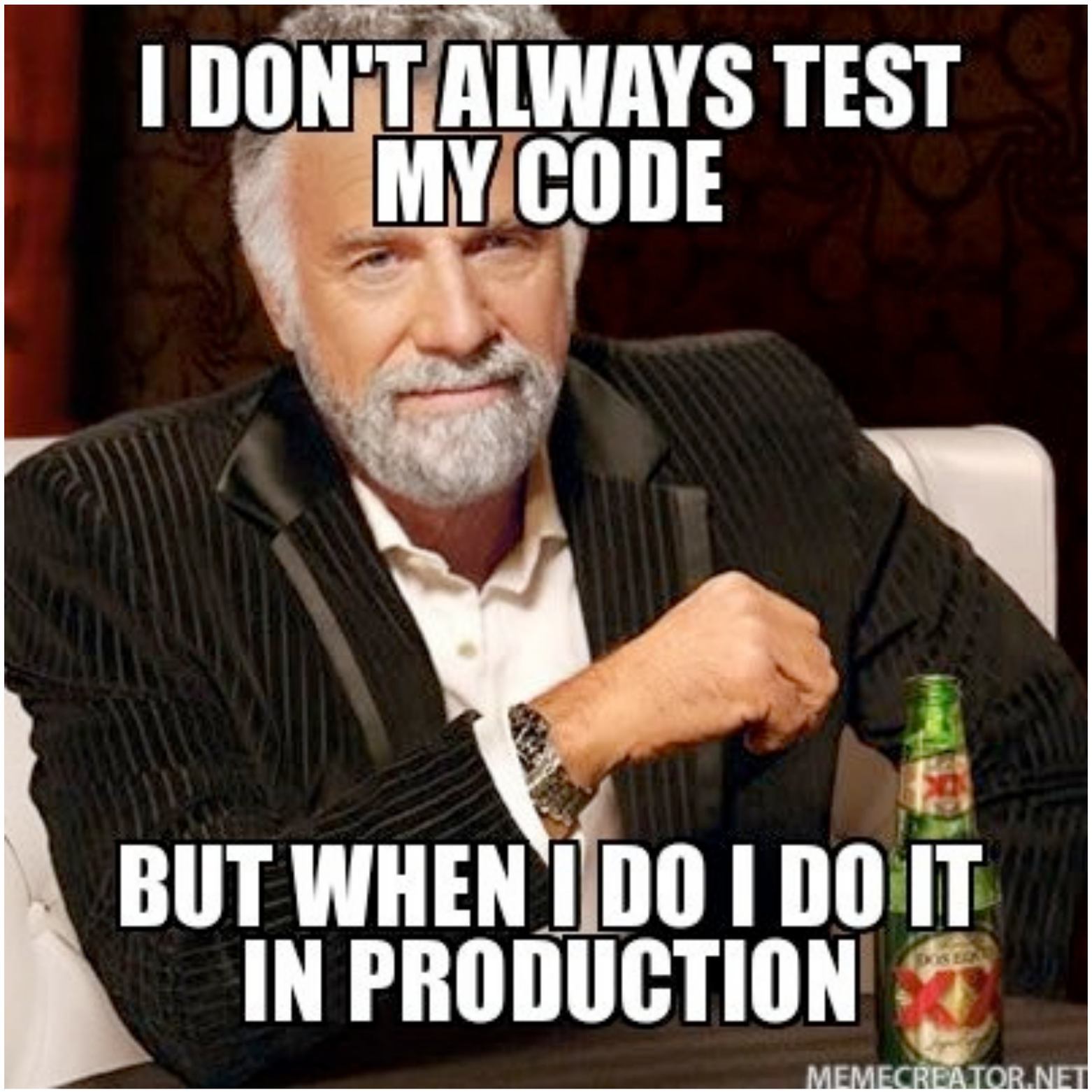
I DON'T ALWAYS TEST
MY CODE
BUT WHEN I DO I DO IT
IN PRODUCTION
MEMECREATOR.NET

Based on materials by Katy Huff, Rachel Slaybaugh, and Anthony Scopatz

https://github.com/swcarpentry/boot-camps/tree/master/python/testing

Outline

- What is a test?
- Why testing?
- Where to put the tests and run them?
- When should I test?
- How to write, run and maintain test?
- NumPy specific utilities and test coverage

What is a test?

```
def compute_square(x):
    return x ** 2


def test_compute_square():
    assert compute_square(0) == 0

    assert compute_square(1) == 1
    assert compute_square(-1) == 1

    assert compute_square(2) == 4
    assert compute_square(-2) == 4
```

Why testing?

- To **validate code behavior** (meet expectations) for many input cases
- To **find bugs earlier** when easy to fix
- To **prevent silent regressions** when refactoring
- To **guide the development** (TDD)
- To **keep the developers motivated**

Types of Tests

- **unit tests**: one function / class at a time
- **integration tests**: many assembly
- **non-regression tests**:
  - find a bug: write a test to reproduce and then fix the bug
  - can be unit tests or integration tests

Where to put the tests?

- Put the code in a module (a Python file):

```
mypackage/__init__.py  # empty
mypackage/mymodule.py
```

- Put the tests in a side module, for instance:

```
mypackage/test_mymodule.py
```

Example

mypackage/mymodule.py

```
def compute_square(x):
    return x ** 2
```

mypackage/test_mymodule.py

```
from mypackage.mymodule import compute_square

def test_compute_square():
    assert compute_square(0) == 0

    assert compute_square(1) == 1
    assert compute_square(-1) == 1

    assert compute_square(2) == 4
    assert compute_square(-2) == 4
```

How to run tests?

```
$ pip install nose                    # or: easy_install nose
Downloading/unpacking nose
  Downloading nose-1.3.0.tar.gz (404kB): 404kB downloaded
  ...
Successfully installed nose
Cleaning up...
```

```
$ nosetests mypackage
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

https://nose.readthedocs.org/

Failures

```
$ nosetests mypackage
F
======================================================================
FAIL: mypackage.test_mymodule.test_compute_square
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/usr/local/lib/python2.7/site-packages/nose/case.py", line 197, in runTest
    self.test(*self.arg)
  File "/Users/ogrisel/coding/scbc-testing-doc-packaging/testing/examples/
mypackage/test_mymodule.py", line 11, in test_compute_square
    assert compute_square(-2) == 3
AssertionError

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

When should I write & run tests?

- As early as possible (TDD)
- As often as possible
- Before every git push to a public repo
- Before fixing a bug (non-regression)

- Tests should be fast to run!

Exercise 01

https://github.com/ogrisel/scbc-testing-doc-packaging/tree/master/testing/exercises/01

nose.tools assertions

- The Python `assert` builtin does not yield very useful error message
- Better `nose.tools.assert_*`
  - `assert_equals(a, b)`
  - `assert_true(x) / assert_false(y)`
  - `assert_in(item, sequence)`

Test Corner Cases

- How should that function react when passed: `None`, zero or negative numbers, empty strings, empty files, NaN inputs...?
- Test the type of exceptions raised in case of invalid input:
  - Wrong type should raise `TypeError`
  - Invalid type should raise `ValueError`

Testing for expected exceptions

```
from nose.tools import assert_raises
from nose.tools import assert_equals


def mean(x):
    if len(x) == 0:
        raise ValueError(
            "mean of empty list is undefined.")
    return float(sum(x)) / len(x)


def test_mean():
    assert_equals(mean([1, 2, 3]), 2)
    assert_equals(mean([-4, 4]), 0)

    assert_raises(ValueError, mean, [])
```

Testing for exceptions and error message

```
from nose.tools import assert_in


def assert_raise_message(exception, message, function,
    *args, **kwargs):
    """Helper to test error messages in exceptions"""
    try:
        function(*args, **kwargs)
        raise AssertionError("Should have raised %r"
                             % exception(message))
    except exception as e:
        error_message = str(e)
        assert_in(message, error_message)
```

Exercise 02

- https://github.com/ogrisel/scbc-testing-doc-packaging/tree/master/testing/exercises/02

Test Coverage

- Count the % of lines that are executed when running the tests.

```
$ nosetests --with-cover --cover-package=scbctesting
...
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
scbctesting                       0      0   100%
scbctesting.nonlinear             5      0   100%
-----------------------------------------------------------
TOTAL                             5      0   100%
-----------------------------------------------------------
Ran 3 tests in 0.011s

OK
```

Exercise 03

- Compute the HTML report for the test coverage of the previous exercise
- Which lines are not covered?
- Can you write a new test to increase the coverage?

Test Fixtures

- Very useful for **Integration Tests**
- Setup components to test integration with
- Examples:
  - Create a temporary file / folder
  - Create a database with test users
  - Open a connection to the test database

Fixtures Execution

```
try:
    setup()
    test_something_1()
finally:
    teardown()
```

```
try:
    setup()
    test_something_2()
finally:
    teardown()

try:
    setup()
    test_something_3()
finally:
    teardown()
```

```python
import unittest, tempfile, csv
import numpy as np
from nose.tools import assert_equals
from numpy.testing import assert_array_almost_equal


class CSVParsingTestCase(unittest.TestCase):

    def setUp(self):
        self.f = tempfile.TemporaryFile()
        self.f.write('0.1,0.2\n')
        self.f.write('-0.1,1.4\n')
        self.f.seek(0)

    def tearDown(self):
        self.f.close()

    def test_numpy_csv(self):
        a = np.loadtxt(self.f, delimiter=',')
        assert a.shape == (2, 2)

        expected = [[0.1, 0.2], [-0.1, 1.4]]
        assert_array_almost_equal(a, expected, 2)

    def test_csv(self):
        dicts = list(csv.DictReader(self.f, fieldnames=['a', 'b']))
        assert len(dicts) == 2

        expected = [{'a': '0.1', 'b': '0.2'}, {'a': '-0.1', 'b': '1.4'}]
        assert_equals(dicts, expected)
```

Demo: debugging tests

- nosetests --pdb --pdb-failures

- pip install ipdbplugin
- nosetests --ipdb --ipdb-failures