

Unit testing in Python

David J. Birnbaum

<http://www.obdurodon.org>

<https://github.com/djbpitt>

djbpitt@gmail.com

PyLing, 2019-03-27

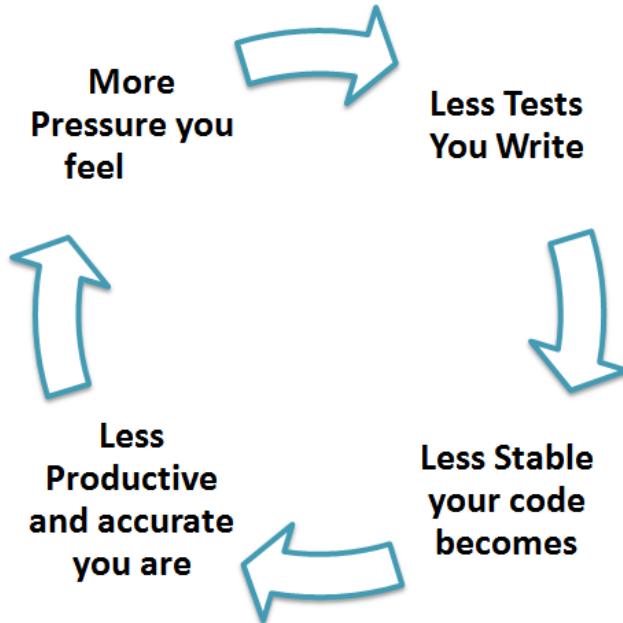
Outline

- About testing
 - Why test
 - Types of testing
- About unit testing
 - What to test
 - Edge cases
 - Variable output
- When to test
- Nose tests
 - Basics
 - Fixtures
- Testing on the command line
- Coverage testing
- Testing inside PyCharm
- Testing on GitHub with Travis CI (continuous integration)
 - With badges

What is testing?

- Find logical errors in your code
- The code runs, but ...
 - It raises an error that you didn't anticipate
 - You calculate an average ... and divide by zero
 - You calculate an average ... and your user inputs text instead of numbers
 - Even worse: it produces the wrong result and you don't notice
 - You divide two numbers and get integer (floor) division when you want a float, or vice versa
 - You compare text strings without attending to whitespace or Unicode normalization

Why people avoid testing



- It gets tested when I run it!
 - Only on the input you give it
 - Only at a high level
- My code is bullet-proof!
 - Er ... no, it isn't ...
- It takes too much time!
 - Not testing takes more time

Types of testing

- Unit testing
 - Smallest testable unit (function, method)
- Integration testing
 - I/O interaction between and among units
- System testing
 - Functionality as a whole
- Acceptance testing
 - End-user testing for suitability to task

What is unit testing?

- Smallest testable code component, in isolation
 - Function, method
- Isolate and test the unit independently of other code
 - Not dependent on filesystem, I/O, other processes, etc.
 - Tests are independent of one another
- If a unit test fails, you know where the error lies
- Tests are run as a suite
 - Framework, harness
 - The entire suite should complete, even if some individual tests fail
 - Testing may be run manually or automated
 - Testing has to be easy, or we won't do it

Digression: what is a pipeline?

```
grep -i 'horrible' review.json | cut -d, -f4 | sort | uniq -c | sort -nr
```

- grep find all lines that contain 'horrible' (case-insensitive)
- cut isolate part of the data
 - -d, means that the delimiter is a comma
 - -f4 means fourth field
- sort sort only fourth field values
- uniq -c collect adjacent lines and put the count in front
- sort numeric sort from high to low
- Tada!

The pipeline

```
def transliterate(line: str) -> str:  
    """  
        Transliterate input line from Cyrillic XML to Roman string  
  
        Keyword parameters:  
        line (str): a line of verse as well-formed XML  
  
        Returns:  
        str: transliterated line  
    """  
    return functools.reduce(  
        lambda value, function: function(value),  
        (  
            _flatten,  
            _lexical,  
            _ogo,  
            _proclitics,  
            _enclitics,  
            _tsa,  
            _palatalize,  
            _jot,  
            _romanize,  
            _final_devoice,  
            _regressive_devoice,  
            _regressive_voice,  
            _regressive_palatalization,  
            _consonant_cleanup,  
            _vowel_reduction  
        ),  
        line,  
    )
```

What to test

- Functionality that can be isolated (functions, methods)
 - Create complex functions as pipelines, not monoliths
- All possible types of input ...
- ... including edge cases

The Edge Case Saloon

- A developer walks into a bar and ...
 - Orders a beer.
 - Orders 0 beers.
 - Orders 999999999 beers.
 - Orders a lizard.
 - Orders -1 beers.
 - Orders a sfdeljknesv.

Sample unit test

```
from nose.tools import *
from cyr2phon import cyr2phon

def test_ого_моего():
    expected = "моев0"
    assert_equal(cyr2phon._ого("моег0"), expected)
```

Testing with variable output

- We can't use `assert_equal()` with, e.g., dictionary output because the order of dictionary entries is undefined
- So find another way

When to test?

- Test as you code
 - When you create code, add tests
 - When you fix a bug, add tests
- Test-driven development
 - Write the tests first, and then add code so that they pass
- Regression testing
 - Run the full test suite often

Nose tests



- Nose is a Python testing framework
 - There are others
- May be run from the command line, within an IDE, or in CI
- May be object oriented or not (mine is not)

Testing involves

- Discovering tests
 - Files or directories under the current working directory whose names include “test” or “Test” at a word boundary (like “test_this” or “functional_test” or “TestClass” but not “libtest”)
- Running tests
- Reporting results

Sample nose test

```
from nose.tools import *
from cyr2phon import cyr2phon

def test_ого_moego():
    expected = "моев0"
    assert_equal(cyr2phon._ого("моег0"), expected)
```

Test generator

```
def test_palatalize_front_vowel():
    for v in "яенёюЯЕИЁ":
        fn = lambda: check_palatalize_front_vowel(v)
        fn.description =
"cyr2phon.tests.test_transliterate.test_palatalize_front_vowel with
{}".format(v)
        yield fn

def check_palatalize_front_vowel(v):
    assert_equal(cyr2phon._palatalize("б" + v), "Б" + v)
```

Fixtures

- Start-up
- Tear-down
- (I don't use these)

Command line

nosetests

-v

--with-coverage

--cover-package=cyr2phon

--cover-tests cyr2phon/tests/test_transliterate.py

--cover-tests cyr2phon/tests/test_utility.py

Coverage testing

- Report about amount of the code base that is addressed by tests

Name	Stmts	Miss	Cover

cyr2phon/__init__.py	6	0	100%
cyr2phon/cyr2phon.py	112	0	100%
cyr2phon/tests/__init__.py	0	0	100%
cyr2phon/tests/test_transliterate.py	172	0	100%
cyr2phon/tests/test_utility.py	24	0	100%
cyr2phon/utility.py	32	0	100%

TOTAL	346	0	100%

Testing inside PyCharm

The screenshot shows the PyCharm IDE interface with the following details:

- Project View:** Shows the file structure of the `russian_rhyme` project, including subfolders like `russian_rhyme`, `cyr2phon`, and `tests`. Inside `tests`, there are files `__init__.py`, `cyr2phon.py`, `lexical.json`, and `utility.py`.
- Code Editor:** Displays the `cyr2phon.py` file with code related to phonetic conversion and normalization.
- Run Tool Window:** Shows the output of a test run:
 - Tests passed: 148 of 148 tests – 43 ms
 - Test Results 43ms
 - Testing started at 16:10 ...
/Users/djb/repos/russian_rhyme/venv/bin/python "/Users/djb/Library/Application Support/JetBrains/Pycharm.app/Contents/bin/nosetests" --with-coverage --cover-package=cyr2phon
 - Ran 148 tests in 0.196s
 - OK
 - Process finished with exit code 0
- Event Log:** Displays log entries from 2019-03-27 at 16:05, indicating that no R interpreter was defined.
- Status Bar:** Shows the current configuration (Run selected configuration with coverage enabled), file encoding (UTF-8), and Git status (master).

Testing inside PyCharm with coverage

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The left sidebar displays the project structure under "russian_rhyme". The "cyr2phon" directory contains several files: .py, .ipynb, and .md. The "tests" directory is highlighted in yellow.
- Code Editor:** The main editor window shows the content of `cyr2phon.py`. The code implements various phonetic functions like `_convert_tsa`, `_palatalize`, `_jot`, `_romanize`, `_final_devoice`, and `_regressive_devoicing`.
- Coverage Analysis:** A floating window titled "Nose tests in cyr2phon" provides coverage statistics:
 - Coverage: 83% files, 99% lines covered
 - Element: .idea
 - Statistics, %: 100% file...
- Run Tab:** The bottom-left tab bar shows the "Run" tab is active, displaying the output of the nose test run.
- Event Log:** The bottom-right panel shows the event log from March 27, 2019, at 16:10, reporting 148 tests passed.
- Status Bar:** The bottom status bar shows the current file is "russian_rhyme (2)" and the encoding is "UTF-8".

Testing on GitHub with Travis CI

- Continuous integration
- Configure `.travis.yml` in repo
- List requirements in `requirements.txt`
- Project “builds” automatically on every push

Travis CI files

.travis.yml

```
language: python
python: 3.7
dist: xenial
sudo: true
script: "nosetests -v --with-coverage --cover-package=cyr2phon --cover-tests cyr2phon/tests/"
after_success: "codecov"
```

requirements.txt

```
numpy
pandas
nose
codecov
coverage
regex
```

GitHub badges

The screenshot shows a GitHub README.md page with the following content:

README.md

monotreme obduronon build passing codecov 99%

Machine-assisted identification of rhyme in Russian verse

David J. Birnbaum

Files and directories

Code

Notebook

- Progress report 1 notebook

Library

- cyr2phon.py*: Cyrillic to phonetic library; exposes `cyr2phon.transliterate()`
- utility.py*: Utility functions; exposes `utility.syllabify()`

Development

- test_transliterate*: Nose tests for [transliteration code](#)
- test_utility*: Nose tests for [utility code](#) (syllabification)
- .travis.yml* and *requirements.txt*: Configuration information for [Travis CI](#)

Badge code

- `[![Obdurodon](images/monotreme-obdurodon-blue.svg)](http://www.obdurodon.org)`
- `[![Build Status](https://travis-ci.com/Data-Science-for-Linguists-2019/russian_rhyme.svg?branch=master)](https://travis-ci.com/Data-Science-for-Linguists-2019/russian_rhyme)`
- `[![Code Coverage](https://codecov.io/gh/Data-Science-for-Linguists-2019/russian_rhyme/branch/master/graph/badge.svg)](https://codecov.io/gh/Data-Science-for-Linguists-2019/russian_rhyme)`

monotreme obdurodon build passing

codecov 99%

Your own badges

- Create an account at <https://travis-ci.com>
 - Connect to repo
- Create an account at <https://codecov.io>
 - Connect to repo
- Download images from <https://shields.io/> or <https://github.com/badges/shields>
- Add links to README.md (see above)

Build report

Travis CI  Dashboard Changelog Documentation Help 

Data-Science-for-Linguists-2019 / russian_rhyme build passing

Current Branches Build History Pull Requests More options 

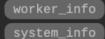
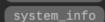
✓ master Merge branch 'feature-matrix-xml-to-json'
-o #96 passed
🕒 Ran for 30 sec
📅 about a month ago

-o Commit 1b9dea5 
Compare e6d59df..1b9dea5 
Branch master 
 djbpitt

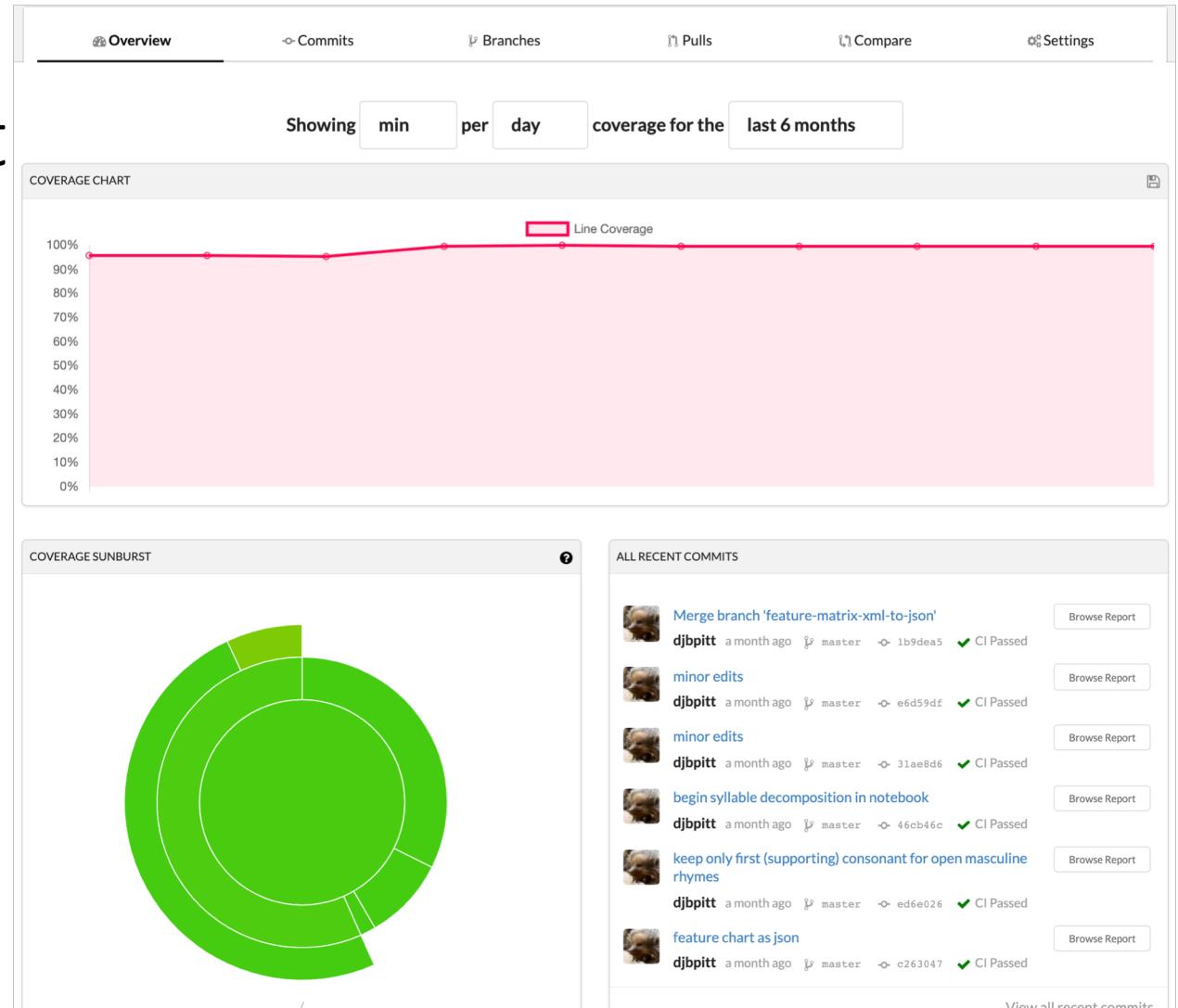
 Python: 3.7

[Job log](#) [View config](#)

X Remove log  

▶ 1 Worker information 
▶ 6 Build system information 
156
157
▶ 158 \$ git clone --depth=50 --branch=master https://github.com/Data-Science-for-Linguists-  0.62s
168
169 \$ source ~/virtualenv/python3.7/bin/activate
170 \$ python --version
171 Python 3.7.1
172 \$ pip --version
173 pip 18.1 from /home/travis/virtualenv/python3.7.1/lib/python3.7/site-packages/pip (python 3.7)
▶ 174 \$ pip install -r requirements.txt  18.89s
0.01s
214 \$ nosetests -v --with-coverage --cover-package=cyr2phon --cover-tests cyr2phon/tests/
215 cyr2phon.tests.test_transliterate.test_PUNC_RE ... ok
0.53s
216 cyr2phon.tests.test_transliterate.test_OGO_BE_with_ora ... ok

Coverage report



Thank you

David J. Birnbaum

<http://www.obdurodon.org>

<https://github.com/djbritt>

djbritt@gmail.com



For more information

- Install: <https://pypi.org/project/nose/1.3.7/>
- Documentation: <https://nose.readthedocs.io/en/latest/index.html>
- Tutorials
 - https://www.tutorialspoint.com/unittest_framework/nose_testing_framework.htm
 - <http://pythontesting.net/framework/nose/nose-introduction/>
 - <http://ivory.idyll.org/articles/nose-intro.html>