Testing in Style - Reducing Barriers to Higher Code Quality Through Automation

Moritz E. Beber

Postdoc in Computational Biology

Novo Nordisk Foundation Center for Biosustainability

Why Test at All?

- The more **problems caught** in development, the less your users (you!) see in production
- High test coverage gives you **confidence** to refactor code
- Similarly, it makes it easier for others to **contribute** because they easily know whether or not they broke something
- Writing testable code typically leads to more **modular** code

Aspects of Testing

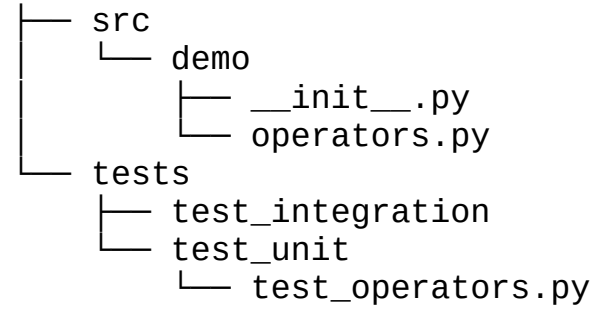
- Follow a style (pep8 (https://www.python.org/dev/peps/pep-0008/))
- Statically analyze your code
- Unit tests
- Integration tests

Live Demo

Contrived example: a simple calculator

```
git clone https://github.com/Midnighter/calculator-demo
```

Package Structure



Why `src`?

- It can get messy
- https://blog.ionelmc.ro/2014/05/25/python-packaging/#the-structure (https://blog.ionelmc.ro/2014/05/25/python-packaging/#the-structure)
- http://andrewsforge.com/article/python-new-package-landscape/ (http://andrewsforge.com/article/python-new-package-landscape/)

Topics

- `flake8-docstrings`
- `pytest.mark.parametrize`
- `pytest-raises`
- `hypothesis`
- `coverage`

Important Topics not Covered

- fixtures (https://docs.pytest.org/en/latest/fixture.html)
- pdb (https://docs.pytest.org/en/latest/usage.html#dropping-to-pdb-python-debugger-on-failures)
- mocking (https://docs.pytest.org/en/latest/monkeypatch.html)

Invoking Tox

Run all the defined `tox` environments:

```
tox
```

Select a specific environment:

```
tox -e py36
```

If your `tox.ini` defines it, you can provide additional arguments after `--`:

```
commands =
    pytest --cov=demo {posargs: tests}
```

```
tox -e py36 -- --cov-report=html tests
```

Non-installed Projects

If you are testing a project that is not an installable package, you will want to change your `tox.ini`:

```
[tox]
envlist = ...
skipsdist = true

[testenv]
skip_install = true
```

You can then install dependencies normally, e.g., via a requirements file:

```
deps =
   -rrequirements.txt
```

or additionally allow global site-packages in the `[testenv]` section:

```
sitepackages = true
```

Coverage

You can measure coverage locally as specified previously but you can also use online providers which gives you:

- Tracking of coverage over branches and commit histories
- Integration with pull (merge) request checks that require a certain coverage

Coverage Providers

- codecov.io (https://codecov.io/):
    - free for open source [+]
    - easily combines parallel reports [+]
- coveralls.io (https://coveralls.io/):
    - free for open source [+]
    - slightly more work for parallel reports [~]
- code climate (https://codeclimate.com/):
    - free for open source [+]
    - investigates code quality [+]
    - parallel reports require separate storage (such as S3 bucket) [-]
- Landscape (https://landscape.io/):
    -
        - free for open source [+]
    -
        - investigates code quality [+]

Continuous Integration & Deployment

- Travis CI (https://travis-ci.org/)
- Circle CI (https://circleci.com/)
- AppVeyor (https://www.appveyor.com/)
- Jenkins (https://jenkins.io/)
- GitLab CI (https://about.gitlab.com/product/continuous-integration/)

Thank You

For a complete example, please take a look at the full-implementation (https://github.com/Midnighter/calculator-demo/tree/full-implementation) branch.

Feel free to reach out to me:

- midnighter@posteo.net (mailto:midnighter@posteo.net?subject=Python testing)
- Midnighter (https://github.com/Midnighter/)
- @me_beber (https://twitter.com/me_beber)
- Moritz Beber (https://www.linkedin.com/in/moritz-beber-b597a55a/)