How to test and document your Python code

Jean-Claude Passy

ZWE Software Workshop

IMPRS-IS S4 Workshop, October 28, 2021

MAX PLANCK INSTITUTE
FOR INTELLIGENT SYSTEMS

Outline

Outline

Types of testing

Testing is very important. It helps you to write better code and save time in the long run. It also protects your reputation. There are many types of testing:

- Unit testing: testing an individual component/functionality
- Integration testing: testing components grouped together
- Functional testing: testing the generated output (black-box)
- Acceptance/validation testing: testing outputs against requirements
- Alpha testing: Testing by developers before release
- Beta testing: Testing by customers before release
- ...

Writing tests in Python

Standard tools to use

- Python: unittest, pytest, nose
- Web applications: Selenium

How do you know you need to write test?

- Code coverage (e.g. coverage.py)
- Find bugs/unexpected usage (increase test coverage)

Some general advice:

- Start writing tests very early on (almost TDD)
- Benchmarks are very useful for projects you are taking over

How?

From now on, we use unittest, the standard framework for testing Python code.

- Create `tests` packages
- Create modules that contains the tests with appropriate names.
- By default, the automatic test discovery expects test modules to be named `"test*.py"`
- Create a test case, i.e. a class inheriting from `unittest.TestCase`
- Method names starting with `"test"` will be understood as tests
- `setUp`/`teardown` methods are executed before/after each test
- Run for example:

```
$ python -m unittest
$ python -m unittest test_module1 test_module2
$ python -m unittest test_module.TestClass
$ python -m unittest test_module.TestClass.test_method
```

Specfic questions

Q: What about testing in ML?

A: We don't really know! But the suggestions might help:

- Validation for corner cases (small samples, theoretical value)
- Randomization and invariants (e.g. metrics must be positive)
- Benchmarks
- Proper structure and mocking with real data
- Read this article (take a single training step and compare invariants, train different sets of variables and check results are different)

Outline

Why should I do that

Writing documentation is one of the most important stage in software development. Without it, your code will not be used and become obsolete. It is also a way to manage expectations and transfer knowledge.

- README: this is the **bare minimum**

- Proper documentation: main framework for Python is Sphinx.

One large part of a good documentation are `docstrings` (see PEP 257). These are strings as first statement of a module/class/function. These are what you see when using the `?`.

Different styles

There are 2/3 styles for writing documentation:

Google

```
"""
This is an example of Google style.

Args:
    param1: This is the first param.
    param2: This is a second param.

Returns:
    This is a description of what is returned.

Raises:
    KeyError: Raises an exception.
"""
```

reStructured text

```
"""
This is a reST style.

:param param1: this is a first param
:param param2: this is a second param
:returns: this is a description of what is returned
:raises keyError: raises an exception
"""
```

How?

The standard framework for writing documentation for Python code is Sphinx.

- Create `doc` folder
- Install sphinx with pip:
  ```
  $ pip install sphinx
  $ pip install sphinx_bootstrap_theme
  ```
- If you use Google style:
  ```
  $ pip install sphinxcontrib-napoleon
  ```
- Run:
  ```
  $ sphinx-quickstart
  $ make html
  ```
  or
  ```
  $ sphinx-build -b html source_dir build_dir
  ```