LECTURE 0b100000

TESTING AND PYTEST

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- Project 3 due 6pm Central
- Project 4 description later today
- Worksheet 11 was posted yesterday (oops!)
- Quiz 11 is due on Monday, and is quite short

ASSERT

Python's `assert` statement is simple: It is followed by an expression. If that expression is truthy, nothing happens. If it is falsy, the AssertionError exception is raised.

One use for `assert` is during debugging of a program under development: It stops execution if expected conditions are not met.

For example,

```
assert fact(0)==1
```

is mostly equivalent to

```
if not fact(0)==1:
    raise AssertionError()
```

So a good use for `assert` is

```
assert thing_expected_to_always_be_truthy
```

TESTING

In software development, **testing** refers to the process of checking that a program does what it is supposed to do, i.e. that it meets the desired specifications.

Most often this takes the form of code that performs testing in an automated way.

Some software development approaches rely very heavily on testing.

TESTING IN PYTHON

Many options for orchestrating tests.

- unittest (part of the standard library)
- pytest (our focus today)
- nose
- *many others...*

PYTEST

I hope you've installed pytest, a popular Python module for testing. Reminder: These shell commands should install and run pytest for most of you.

```
python -m pip install pytest  # install pytest
python -m pytest              # run pytest
```

PYTEST BASICS

To use pytest, you need some Python source files with names that begin `test_` in which there are functions that begin `test_`.

Each such function should run a single test, returning if it succeeds and raising AssertionError (or another Exception) if it fails.

First example: Let's make tests for `fact`.

RUNNING TESTS

In the directory containing your program, run the shell command

```
python -m pytest
```

to find and run all tests. A helpful report is displayed.

It's best to have just one program in a directory when using pytest.

RECOMMENDED PROCESS

Make tests that encode essential functions of the program, with most of them testing a single part (function, class, etc.) in isolation. These are *unit tests*.

Every time you make a change to the program, run the tests again.

- If all tests pass: Proceed.
- If a test fails: Correct the code or correct the test.

ASKING FOR RESOURCES

Most pytest test functions accept no arguments. But if you write one with an argument called `tmpdir`, pytest will create a temporary directory before the test runs, and delete it when the test ends.

The `tmpdir` argument will be an object that contains data about the created directory. Calling `str(tmpdir)` will give the full path as a string.

This feature is very helpful when a test involves writing and reading files.

REFERENCES

- pytest official documentation
- A pytest tutorial from RealPython.com

REVISION HISTORY

- 2020-11-05 Initial publication