# UNIT TESTS AND CONTINUOUS INTEGRATION

#### **Erik Tollerud**

@eteq
Space Telescope Science Institute
Giacconi Fellow
Astropy Coordination Committee



### WHAT TESTING IS NEEDED FOR SCIENCE CODE?

- Functional testing: "Does it run from beginning to end?" or "Does my example work?"
- Result-oriented testing: "Do I get out the science I expect if I run the whole thing on known inputs?"
- Unit testing: "Does this function give me something sensible when I call it?"

#### UNIT TESTING PRINCIPALS

- Whenever you write a "chunk" of code, write corresponding tests.
  - Modular code is *critical* here.
- Make them as specific and fine-grained as you can.
- When you find a bug, write a "regression test".
- Run them early and often!

```
(SILLY) EXAMPLE
     def silly_walk(step):
         if 'left' in step:
             return 'twirl-right'
         elif step == 'twirl-right':
            return 'right'
         elif step == 'right':
             return 'stomp-left'
         else:
            return 'left'
     step = 'arg'
     step = silly walk(step)
     start = step = silly walk(step)
asser for i in range(3):
                                   left'
         step = silly walk(step)
     assert step == start
```

#### **EMPHASIZE THE ASSERT!!**

Second version catches an error: the "l" in "left" is actually a "one"/1.

#### How do you do this? Use A Testing Framework

Documentation » The Python Standard Library » 26. Development Tools »



26.4. unittest — Unit testing













Both use a very similar naming/using convention:

```
# In the file test_code.py

def test_something():
    assert this_should_be == to_this
```





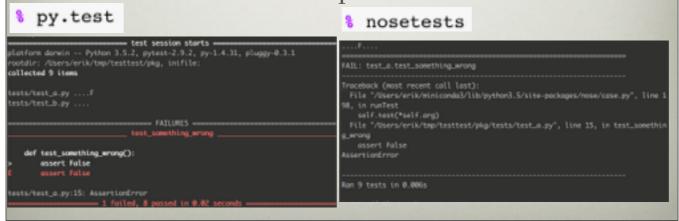
Packages are generally laid out as:

mypackage/\_\_init\_\_.py
mypackage/mymodule.py
mypackage/secondmodule.py
mypackage/tests/\_\_init\_\_.py
mypackage/tests/test\_mymodule.py
mypackage/tests/test\_secondmodule.py
mypackage/tests/test\_some\_use\_case.py





And you can run both very easily with a nice pretty output:



### THE PRIMARY (DIS-?) ADVANTAGE: FLEXIBILITY

#### pytest\_plugins.py

## How Do you Ensure Your Tests Are complete?



(with plugins), works well with





Just remember to check!

## How do you Make Sure you Do Them?

- Consider it part of doing the code.
- You probably do tests anyway. Just write 'em up.
- Adopt test-driven design (although can be hard for science).
- Use continuous integration to make it worth your time.

#### CONTINUOUS INTEGRATION

Run the tests whenever something changes (online).



Especially when done with Pull Requests, this drastically reduces the number of "Oops, sorry, didn't realize that would happen" incidences.

# GO TO: HTTPS://GITHUB.COM/ETEQ/ PYASTRO17-TUTORIALS

GET:
TESTINGANDCI.IPYNB