

Testing

CMSE 890-602

Testing Workflows

- This is something you already do!
 - Fix errors
 - Compare output to expected results
 - Attempt to reproduce other work
 - Show it to colleagues
- But usually informally and manually

Why automate tests?

- Ensure reproducibility
- Reduce bugs
- Catch bugs
- Can develop to tests (“Test driven development”)
 - This is NOT easy for scientific software
- As with last week, a lot of this is a *design choice*
 - But some automated testing is a good idea

Unit testing

- Test individual functions
- Create sets of input and expected output
- “assert” that the function output matches the expected output
- Difficult with floating point numbers!
 - Test frameworks often have “almost equal” assertions with tolerance

Unit testing example (Python)

```
"""
```

```
An example test case with pytest.
```

```
See: https://docs.pytest.org/en/6.2.x/index.html
```

```
"""
```

```
# content of test_sample.py
```

```
def inc(x):
```

```
    return x + 1
```

```
def test_answer():
```

```
    assert inc(3) == 5 # will fail
```

Unit testing example (R)

```
my_good_sum <- function(x, y) x + y

my_buggy_sum <- function(x, y) x + y + 42

# Pass

expect_identical(my_good_sum(6, 4), 10)

# Fail

expect_identical(my_buggy_sum(6, 4), 10)

## Error: my_buggy_sum(6, 4) not identical to 10.

## 1/1 mismatches

## [1] 52 - 10 == 42
```

Unit testing real-world example

https://github.com/tardis-sn/tardis/blob/master/tardis/spectrum/tests/test_spectrum.py

Unit testing workflows

- Workflow managers may have built-in testing capabilities
 - E.g. snakemake can generate unit tests for workflow steps
- BUT this should only be used for “small” (order of MBs) output
- Large outputs are hard to test!
 - See: regression testing

Regression testing

- Test output from code compared to previous output for the same input
- Designed to track changes to overall results
- Often comparisons are between entire tables of data
- May require comparisons of graphical output
 - Difficult! But tools exist.
- Very common in scientific software where unit testing is difficult
- Also called integration testing
- See e.g. <https://astrobarker.github.io/regression-testing-python.html>

Regression testing

- https://github.com/tardis-sn/tardis/blob/master/tardis/tests/test_tardis_full.py

Regression testing workflows

- Workflows may produce large datasets
- Concerns:
 - Storage
 - Bandwidth
 - Computational cost
- May require “mock” workflow runs to produce useful test data
- May require custom software solutions to compare outputs

Regression vs Unit Testing

Regression

- Ensure results are the same as *last time*
- Tests whole code
- Often run on final results
- More useful and common in scientific software

Unit Testing

- Test individual functions (“units”) of code
- May require a change in code to pass
- Best for getting “known” or “intended” results

Testing frameworks

- Simplify test management
- Provide tools for:
 - Grouping tests
 - Configuring tests
 - Handling input/output
- Exist for most programming languages
 - Python: unittest, PyTest
 - R: testthat
 - Java: JUnit, JBehave

Parameterized tests

- Give tests a *set* of input parameters and expected output
- Allows many cases to be tested with one test function
- Handled by testing frameworks (R's `testthat` does not provide parametrization at this time)

```
# content of test_expectation.py
import pytest

@pytest.mark.parametrize("test_input,expected", [("3+5", 8), ("2+4", 6), ("6*9", 42)])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

Testing modules and using fixtures

- Fixtures (contexts) provide objects that can be reused through tests

<https://semaphoreci.com/community/tutorials/testing-python-applications-with-pytest>

Activity

<https://classroom.github.com/a/x0VTKmtv>

- Write unit tests for each function in `example_functions.py`
 - **Remember naming** (`test_`.py and `test_` for function names)
 - At least **2 tested inputs** for each test (ideally using `parametrize`)
- Write functions that pass the tests in `test_examples.py`
 - You can “cheat” the solution to begin with (hardcoding)
 - Work towards the correct functions

Pre-class 4: Documentation

This week the pre-class assignment is to install these packages into a new Python environment:

`mkdocs`

`mkdocstrings-python`

`mkdocs-material`

Then, follow the mkdocs Getting Started here: <https://www.mkdocs.org/getting-started/>

Complete up to “Other Commands and Options”

Upload screenshots to D2L of your docs pages like the screenshots on the Getting Started page.