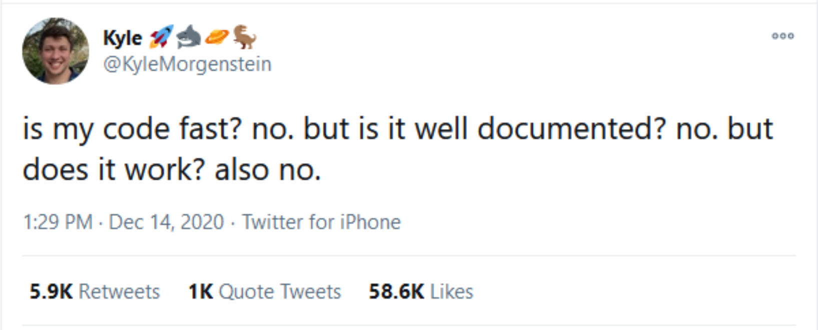
Testing: standalone lecture

Good research code

Patrick Mineault

Intro

Who is this lecture for?



Who is this lecture for?

- Most people who do coding-heavy research are not trained in CS or software engineering
- You're probably in this bucket
- Bad consequences:
  - You feel like you don't know what you're doing
  - Imposter syndrome
  - Low productivity
  - Bugs
  - You hate your code and you don't want to work on it
  - You never graduate
  - You have great sadness in your heart
- It doesn't have to be all bad!

My weird perspective

- Patrick Mineault, PhD in neuroscience
- (wildly underqualified) software engineer at Google
- Research scientist at Facebook on brain-computer interfaces
- Technical chair of Neuromatch Academy
- Independent researcher and technologist
- Occasionally taught CS

Regrets, I've had a few

- Mostly self-taught in programming
- Didn't study CS until very late
- Wasted months working with bad code of my own making
- Not a great coder, but better than in grad school
- I think you might be curious

The single most useful skill

Testing

Organization

- Assume that you know a little bit about Python, git and the command line
  - You can catch up on these topics via Software Carpentries
- I don't expect you to have any experience in packaging code, distribution, working in groups.
- This is a subset of a longer series of lectures which you can refer to
  - https://github.com/patrickmineault/research_code
  - One day I will record the whole lecture set and maybe run it live
- Interrupt me and chat!
- Learning objectives for this lecture
  - What is testing?
  - What should I test?
  - How can I test?
  - How can I integrate testing into a project I'm doing right now?

Bulding around testing

Most scientists who write software constantly test their code. That is, if you are a scientist writing software, I am sure that you have tried to see how well your code works by running every new function you write, examining the inputs and the outputs of the function, to see if the code runs properly (without error), and to see whether the results make sense. Automated code testing takes this informal practice, makes it formal, and automates it, so that you can make sure that your code does what it is supposed to do, even as you go about making changes around it. –Ariel Rokem, Shablona README

Open discussion

- Let's say we have a function in `fib.py`:

```python
def fib(n):
    if n >= 2:
        return fib(n-2) + fib(n-1)
    else:
        return 1
```

- Let's test `fib.py`
- What can we test?

What can we test about `fib`?

- Correctness, e.g. $F(4) = 5$
- Edge cases, e.g. $F(0) = 1$, $F(-1) \rightarrow \mathit{error}$

How can you decide what to test?

- If something caused a bug, test it
  - 70% of bugs will be old bugs that keep reappearing
- If you manually checked if procedure X yielded reasonable results, write a test for it.

What will this give me?

- Decrease bugs: You'll uncover bugs which you'll fix immediately
- Peace of mind: You'll know that your code is correct
- Easy refactors: If you change your code you can easily find out if it's still correct
- Docs: You will know how to call your code long after you've stopped working on it actively
- Better code: If you write your code to be testable you'll write better-organized code

How can we test?

- `assert`
- Hide code behind `if __name__ == '__main__'`
- Test suite

`assert`

- `assert` throws an error if the assertion is False

```
assert -(7 // 2) == (-7 // 2)
```

- Great for inline tests
  - e.g. check whether the shape of a matrix is correct after a permute operation

Hide code behind `if __name__ == '__main__'`

- Code behind `__name__ == '__main__'` is only run if you run the file as a script directly.
- Use this for lightweight tests in combination with `assert`.

```
if __name__ == '__main__':
    assert fib(4) == 5
```

Use a test suite

- Create a specialized file with tests that run with the help of a runner.
- There's `pytest` and `unittest`.
- I use `unittest` because that's what I learned, and it's built-in, but people like `pytest` a lot.

Basic template

```python
# test_something.py
import unittest

class MyTest(unittest.TestCase):
    def sample_test(self):
        self.assertTrue(True)

if __name__ == '__main__':
    unittest.main()
```

Run it

```
$ python test_something.py
```

To run all tests within a directory, install nose via `pip install nose2`, then:

```
$ nose2
```

Live coding

Let's code up `fib.py` tests!

Points from live coding example

- Paths!
    - Sometimes you can get away with hacking `sys.path`
    - Ideally, set up a package with `pip install -e .`
- There's a lot of cruft in writing tests: no shame in copy and paste (but do it once from scratch)!

A hierarchy of tests can be run with a runner

- Static tests (literally your editor parsing your code to figure out if it will crash)
- Asserts
- Unit tests (test one function = one unit; what we just saw)
- Integration tests
- Smoke tests (does it crash?)
- Regression tests
- E2E (literally a robot clicking buttons)

Write lots of tiny unit tests that run very quickly

- Goal: each unit test should run in 1 ms.
- The faster you iterate, the better
  - If your test suite takes more than 5 seconds to run, you will be tempted to go do something else.

Open discussion

Q: what do you think is the ratio of test code to real code in a real codebase?

Open discussion

A: 1:1 to 3:1, but can be many, many times that in safety critical applications

e.g. the aviation standard DO-178C requires 100% code coverage (percentage of lines of code called by the tests) at its third highest safety level (Level C).

For more down-to-earth applications, 80% code coverage is a common target. You can use the `Coverage.py` package to figure out your test coverage.

Demo

Let's code up a non-trivial set of tests for a real paper.

Background on centered kernel alignment

Q: How can we compare how different brain areas and artificial neural networks represent the world?

A: Choose a standard battery of stimuli, measure responses across systems, compare the responses between the systems. Many approaches, including:

- forward encoding models (e.g. ridge regression)
- canonical correlation analysis (CCA)
- representational similarity analysis (RSA).

CKA

Kornblith et al. (2019) propose a new method to compare representations. You can think of it as a generalization of the (square of the) Pearson correlation coefficient, but with matrices instead of vectors.

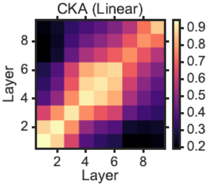


Figure 1: Alignment between layers of two neural nets initialized with different seeds

Importantly, CKA is not implemented in scipy or sklearn, github gives very few hits [1]... it's real research code!

What we know about CKA

- Pearson correlation: If $\mathbf{X}$ and $\mathbf{Y}$ are one-dimensional, then $CKA = \rho(\mathbf{X}, \mathbf{Y})^2$.
- Only makes sense if two matrices are the same size along the first dimension
- $CKA(\mathbf{X}, \mathbf{X}) = 1$

Live coding

Note: to follow at home, look at `cka_step3.py` and `tests/test_cka_step3.py`.

Points from live coding example

- Your test code can be ugly, as long as it's functional!
- Define boundary conditions, pathological examples
  - Test that bad inputs indeed raise errors! Your code should yell when you feed it bad inputs.
- Lock in current behaviour for regression testing
  - E.g. we implement a different, faster implementation of CKA in `cka_step4.py` and regression test it in `test_cka_step4.py`.

Refactoring with confidence

- Your code is ugly: time to refactor!
  1. Your code is ugly, tests pass
  2. Rewrite the code
  3. Your code is clean, tests don't pass
  4. Rewrite the code
  5. Iterate until tests pass again
- Much less stressful with tests and git
- Focus on one test at a time with `python test_cka_step3.py TestCka.test_same`
  - Don't forget to run the whole suite at the end!

Advanced topics!

Testing deterministic side-effect free computational code has a very high returns:effort ratio, but...

- You can also test data loaders for correctness.
- You can also test data for correctness
- You can also test notebooks for correctness
- You can integrate your tests into Github
  - This presentation's repo has CI! It's completely unnecessary!
- You can test stochastic functions

Lesson 3

- Test your code
- Your 5-minute assignment: find a commented-out `print` statement in your code and replace it with `assert`