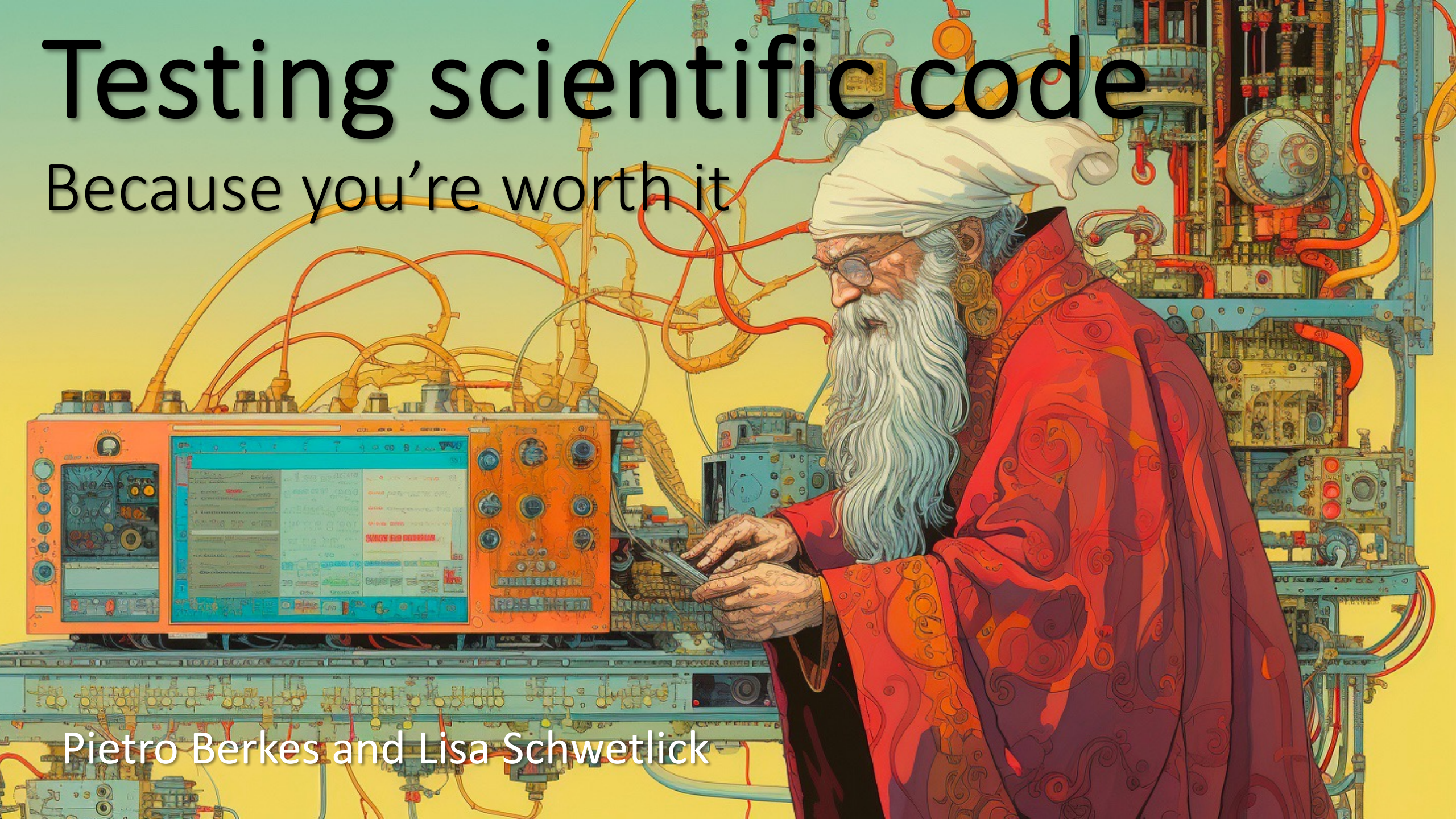
Testing scientific code

Because you're worth it



Pietro Berkes and Lisa Schwetlick

You, as the Master of Research

You start a new project and identify a number of possible leads.

You **quickly develop a prototype** of the most promising ones; once a prototype is finished, you can **confidently decide** whether it is is a dead end, or worth pursuing.

Once you find an idea on which it is worth spending energy, you take the prototype and **easily re-organize and optimize it** so that it scales up to the full size of your problem.

As expected, the scaled-up experiment delivers good results, and your next paper is under way.

How to reach enlightenment

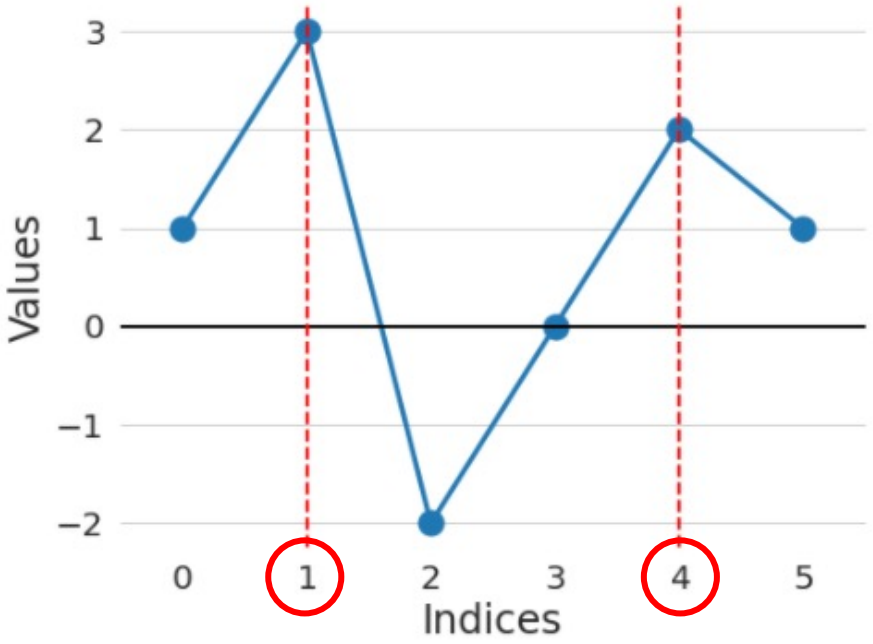
- How do we get to the blessed state of **confidence** and **efficiency**?
- Being a Python expert is not sufficient, good programming practices make a big difference
- We can learn a lot from the development methods developed for commercial and open source software

Outline

- The agile programming cycle
- Testing scientific code basics
- Testing patterns for scientific code
- Continuous Integration

Warm-up project

- Go to the directory called `hands_on/local_maxima`
- In the file called `local_maxima.py`, write a function `find_maxima` that finds the indices of local maxima in a list of numbers



For example,
`find_maxima([1, 3, -2, 0, 2, 1])`
should return
`[1, 4]`

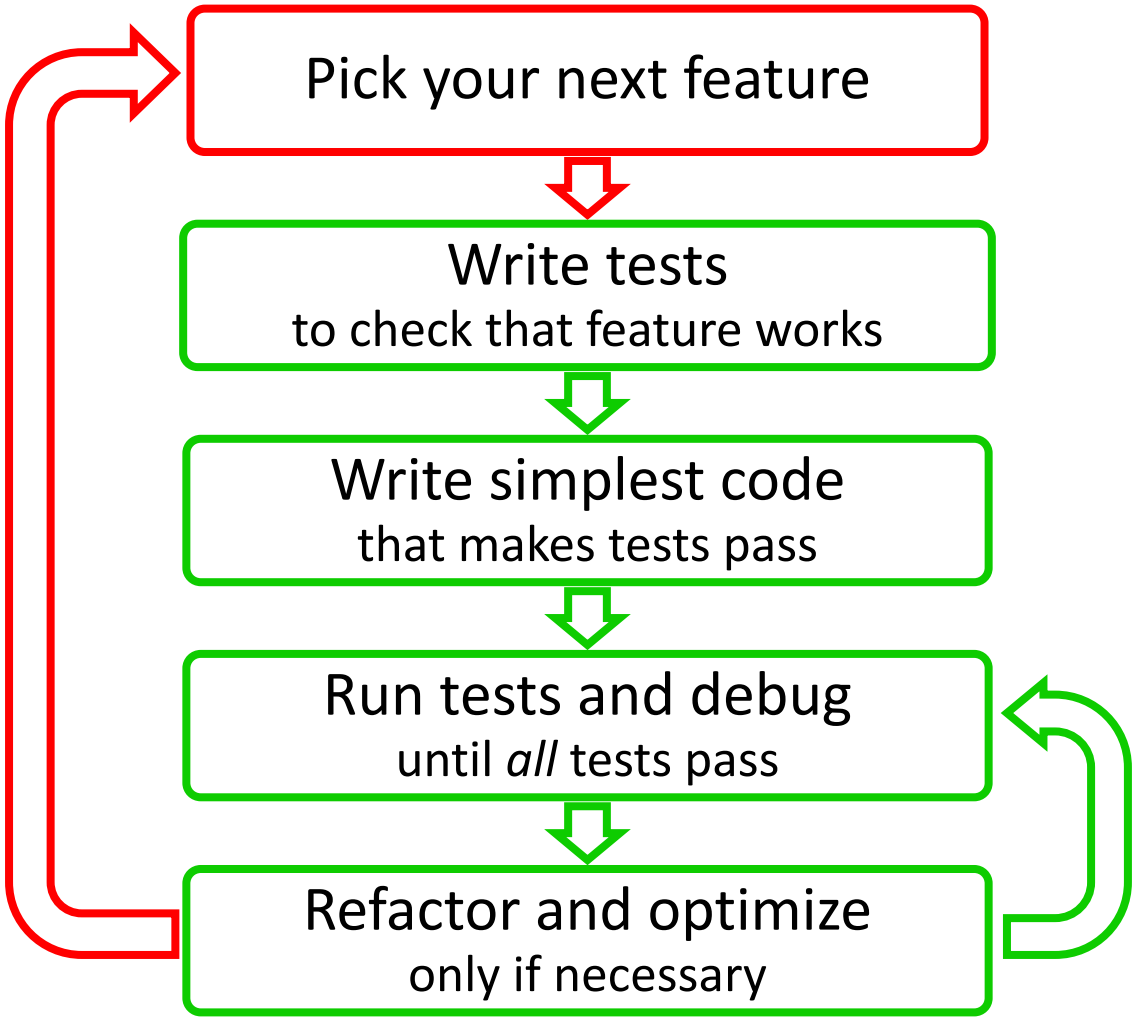
Warm-up project

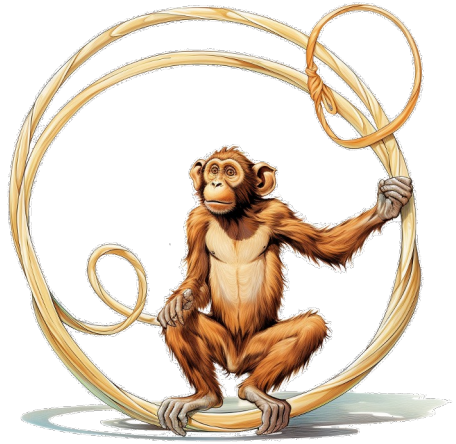
- Write a function `find_maxima` that finds the indices of local maxima in a list of numbers

- Check your solution with these inputs:
  - Input: [1, 3, -2, 0, 2, 1] Expected result: [1, 4]
  - Input: [4, 2, 1, 3, 1, 5] Expected result: [0, 3, 5]
  - Input: [] Expected result: []
  - Input: [1, 2, 2, 1] Expected result: [1] (or [2], or [1, 2])
  - Input: [1, 2, 2, 3, 1] Expected result: [3]

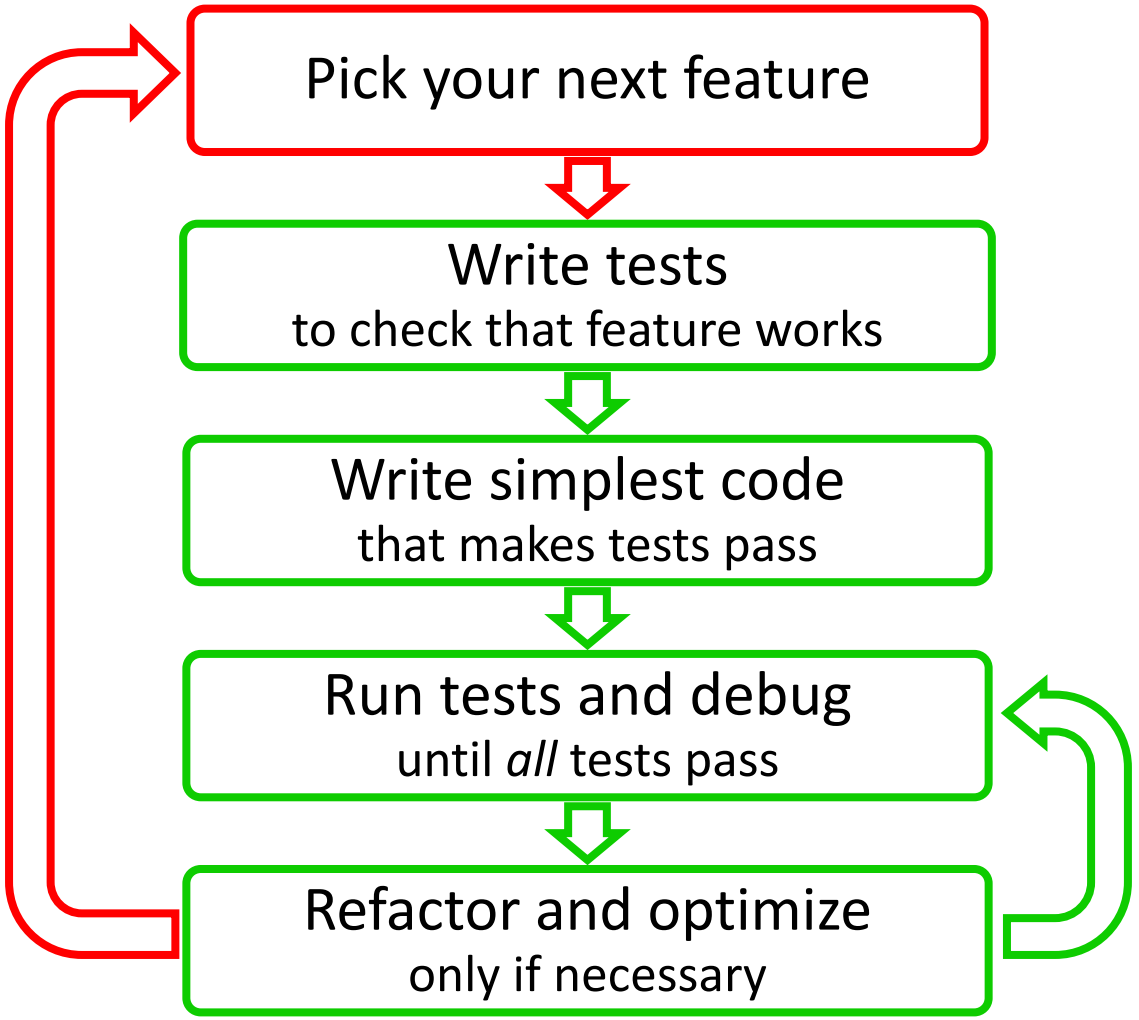
The agile programming cycle

The agile development cycle





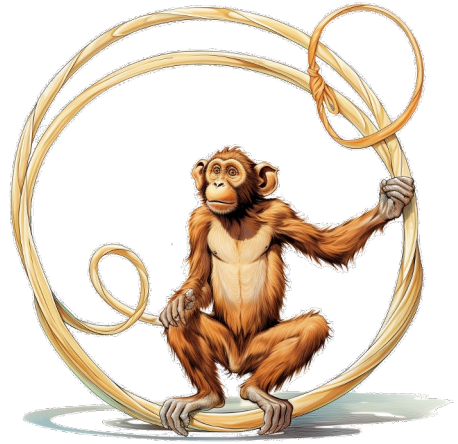
Python tools for agile development



`pytest`

`pdb`

`timeit`
`cProfile`
`line_profiler`

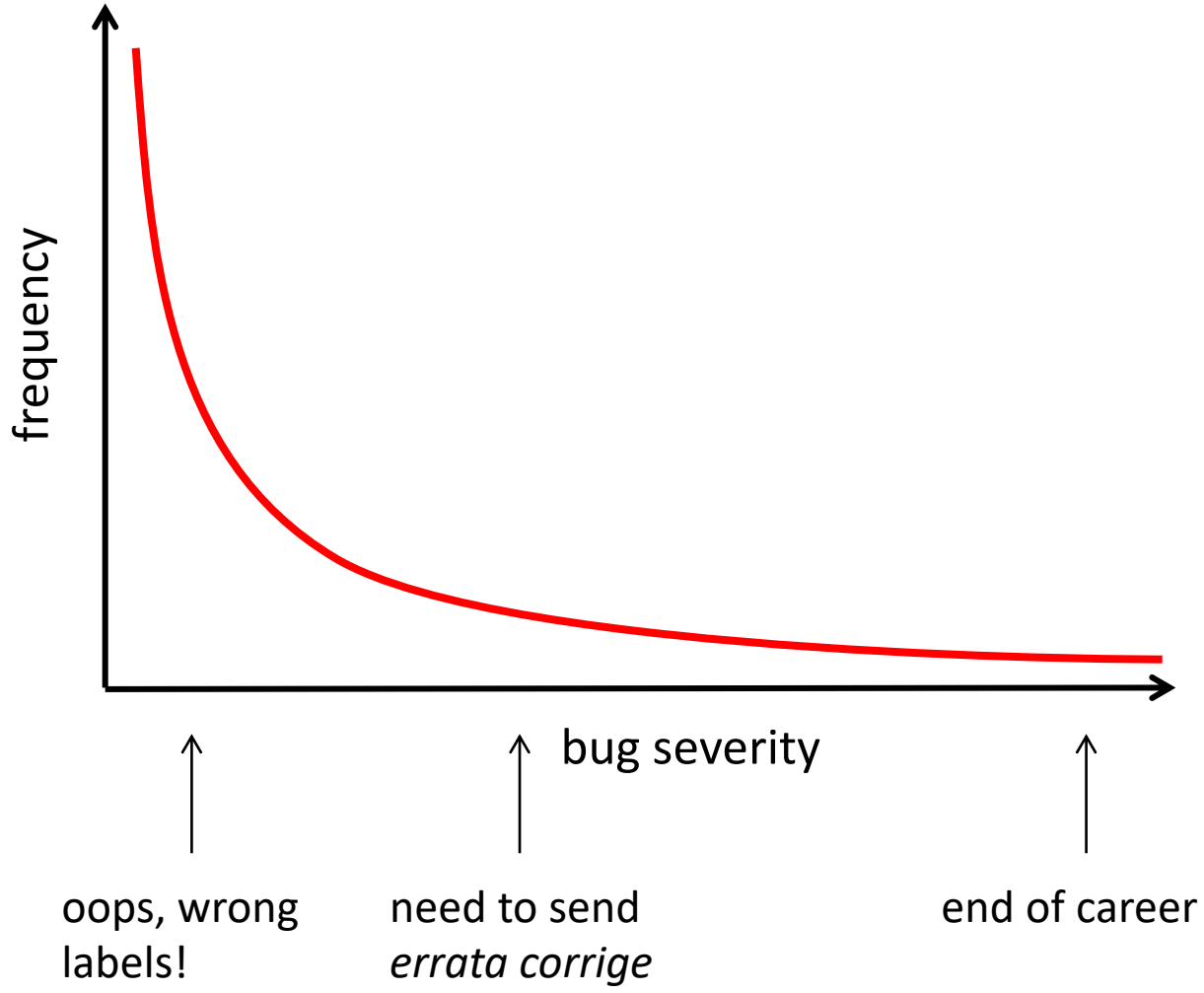


Why test scientific code?

Why write tests at all?

- **Confidence:**
  - Write the code once and use it confidently everywhere else:
    avoid the *negative result* effect!
- **Correctness:**
  - Correctness is main requirement for scientific code
  - You **must** have a strategy to ensure correctness
- **Flexibility:**
  - When you have tests, you can change any part of your code and you will be sure the rest continues to work

Effect of software bugs in science



The unfortunate story of Geoffrey Chang

Science, Dec 2006: 5 high-profile retractions (3x Science, PNAS, J. Mol. Biol.) because ”an in-house data reduction program introduced a change in sign for anomalous differences”

SCIENTIFIC PUBLISHING

A Scientist's Nightmare: Software Problem Leads to Five Retractions

Until recently, Geoffrey Chang's career was on a trajectory most young scientists only dream about. In 1999, at the age of 28, the protein crystallographer landed a faculty position at the prestigious Scripps Research Institute in San Diego, California. The next year, in a cer-

2001 *Science* paper, which described the structure of a protein called MsbA, isolated from the bacterium *Escherichia coli*. MsbA belongs to a huge and ancient family of molecules that use energy from adenosine triphosphate to transport molecules across cell membranes. These

LETTERS

edited by Etta Kavanagh

Retraction

WE WISH TO RETRACT OUR RESEARCH ARTICLE "STRUCTURE OF MsbA from *E. coli*: A homolog of the multidrug resistance ATP binding cassette (ABC) transporters" and both of our Reports "Structure of the ABC transporter MsbA in complex with ADP•vanadate and lipopolysaccharide" and "X-ray structure of the EmrE multidrug transporter in complex with a substrate" (*1*–*3*).

The recently reported structure of Sav1866 (*4*) indicated that our MsbA structures (*1*, *2*, *5*) were incorrect in both the hand of the structure and the topology. Thus, our biological interpretations based on these inverted models for MsbA are invalid.

An in-house data reduction program introduced a change in sign for anomalous differences. This program, which was not part of a conventional data processing package, converted the anomalous pairs (I+ and I–) to (F– and F+), thereby introducing a sign change. As the diffraction data collected for each set of MsbA crystals and for the EmrE crystals were processed with the same program, the structures reported in (*1*–*3*, *5*, *6*) had the wrong hand.

Catastrophic software errors doomed Boeing's airplanes and nearly destroyed its NASA spaceship. Experts blame the leadership's 'lack of engineering culture.'

Morgan McFall-Johnsen | Publié le 29/02/2020 à 14h07

Just 31 minutes after Boeing's CST-100 Starliner spaceship launched into space, Mission Control knew something was wrong.

In the early stages of that crucial test flight on December 20, the Starliner's engines were supposed to fire automatically, setting the ship on a course toward the International Space Station - but they never did.

Mission controllers soon realized the problem: The Starliner's clock was 11 hours ahead. It was following the steps of a phase of the mission it had not yet reached, firing small thrusters to adjust its position.

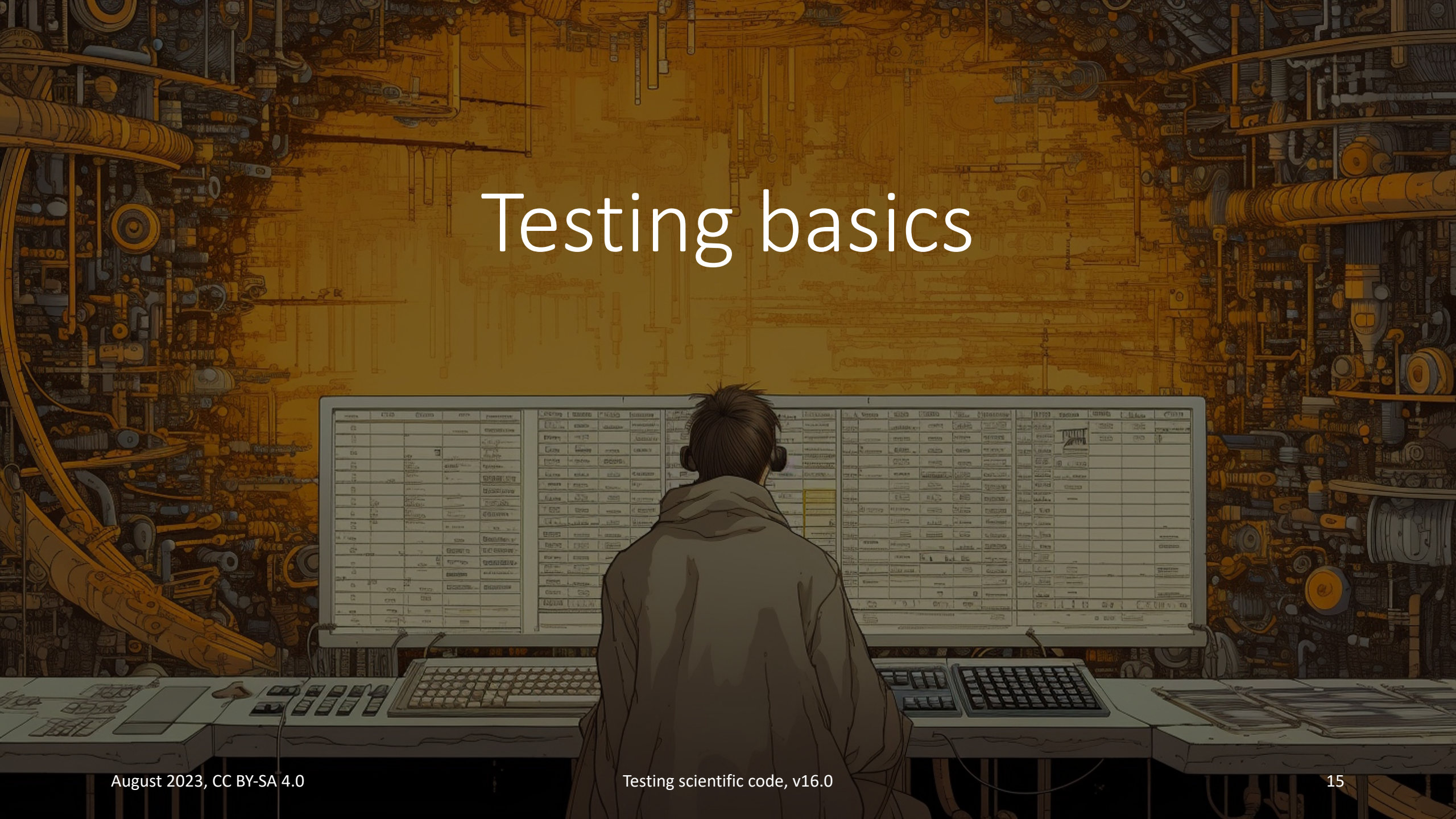
[...]

As Boeing workers frantically checked hundreds of thousands of lines of the spaceship's code, they found a *second* error - one that would have caused the wrong thrusters to fire after two modules of the spacecraft separated. That could have led to a disastrous collision in space.

The Orlando Sentinel reported on Wednesday that a critical end-to-end software test ahead of the Starliner launch could have caught the two coding errors, but Boeing didn't conduct that test at all.

https://www.businessinsider.fr/us/boeing-software-errors-jeopardized-starliner-spaceship-737-max-planes-2020-2

Testing basics



A test is just another function

- Imagine we wrote this new function, and we wanted to test it

```
def times_3(x):
    """Multiply x by 3.

    Parameters
    ----------
    x : The item to multiply by 3.
    """
    return x * 3
```

Testing frameworks

- The collection of tests written to test a package is called a “test suite”
- Execution of a test suite is automated: external software runs the tests and provides reports and statistics
- Main testing frameworks for python:
  - unittest: in the standard library
  - **pytest: what is most commonly used**

```
============================================= test session starts ==============================================
platform darwin -- Python 3.11.3, pytest-7.3.1, pluggy-1.0.0
collected 2 items

test_first.py::test_times_3_integer PASSED                                                              [ 50%]
test_first.py::test_times_3_string PASSED                                                               [100%]

============================================== 2 passed in 0.00s ===============================================
```

Hands-on!

- Go to `hands_on/first`

1. Discover all tests in all subdirectories
   `pytest -v`
2. Execute all tests in one module
   `pytest -v test_first.py`
3. Execute one single test
   `pytest -v test_first.py::test_times_3_string`

Test suites in Python with pytest

- Writing tests with pytest is simple:
  - Tests are collected in files called `test_abc.py` , which usually contains tests for the functions defined in a corresponding module `abc`
  - Each test is a function called `test_jkl_feature`, and usually it tests feature `feature` of a function called `jkl`
  - Each test tests **one feature** in your code, and checks that it behaves correctly using “assertions”. An exception is raised if it does not work as expected.

Assertions

- `assert` statements check that some condition is met, and raise an exception otherwise
- Check that statement is true/false:

```
assert 'Hi'.islower()      => fail
assert not 'Hi'.islower()  => pass
```

- Check that two objects are equal:

```
assert 2 + 1 == 3              => pass
assert [2] + [1] == [2, 1] => pass
assert 'a' + 'b' != 'ab'   => fail
```

- `assert` can be used to compare all sorts of objects, and pytest will take care of producing an appropriate error message

Hands-on! Possibly your first test

- Inside `test_first.py`, test that `first([1])` returns what you would expect

- Try to change the expected result in the test to `[2]` (or anything you want) and watch the test break. Look at the error message and make sure you understand what's going on

Hands-on!

- Create a new file, `test_plus.py`: test that 1+2 is 3
- Execute the tests

Hands-on!

- Create a new file, `test_plus.py`:
  test that 1+2 is 3
- Execute the tests

- Then write a new test and check that 1.1 + 2.2 is 3.3
- Execute the tests

Floating point equality

- Real numbers are represented approximately as “floating point” numbers. When developing numerical code, we have to allow for approximation errors.
- Check that two numbers are approximately equal:

```
from math import isclose
def test_floating_point_math():
    assert isclose(1.1 + 2.2, 3.3)              => pass
```

- `abs_tol` controls the absolute tolerance:

```
assert isclose(1.121, 1.2, abs_tol=0.1)         => pass
assert isclose(1.121, 1.2, abs_tol=0.01)        => fail
```

- `rel_tol` controls the relative tolerance:

```
assert isclose(120.1, 121.4, rel_tol=0.1)       => pass
assert isclose(120.4, 121.4, rel_tol=0.01)      => fail
```

Hands-on!

- One more equality test: check that the sum of these two NumPy arrays:

```
x = np.array([1, 1])
y = np.array([2, 2])
```

  is equal to

```
z = np.array([3, 3])
```

Testing with numpy arrays

```
def test_numpy_equality():
    x = np.array([1, 1])
    y = np.array([2, 2])
    z = np.array([3, 3])
    assert x + y == z
```

```
_____________________________________ test_numpy_equality _____________________________________

    def test_numpy_equality():
        x = numpy.array([1, 1])
        y = numpy.array([2, 2])
        z = numpy.array([3, 3])
>       assert x + y == z
E       ValueError: The truth value of an array with more than one element is ambiguous.
Use a.any() or a.all()

code.py:47: ValueError
```

Testing with numpy arrays

- The module `np.testing` defines helper functions:

```
assert_equal(x, y)
assert_allclose(x, y, rtol=1e-07, atol=0)
```

- If you need to check more complex conditions:
  - `np.all(x)`: returns True if all elements of x are true
    `np.any(x)`: returns True is any of the elements of x is true
  - combine with `logical_and, logical_or, logical_not`:

```
# test that all elements of x are between 0 and 1
assert all(logical_and(x > 0.0, x < 1.0))
```

Watch out for `nans`!

- In general, `nan` is not equal to itself (IEEE standard)

```
In [2]: np.nan == np.nan
Out[2]: False
```

- `assert_equal` and `assert_allclose` consider `nans` equal by default

```
def test_allclose_with_nan():
    x = np.array([1.1, np.nan])
    y = np.array([2.2, np.nan])
    z = np.array([3.3, np.nan])
    assert_allclose(x + y, z)

test_numpy_equality.py::test_allclose_with_nan      PASSED
```

Write a working version of `find_maxima`, with testing

- **Read carefully the description of Issue #2 on GitHub**
- Submit a Pull Request for Issue #2
  - Fork the repository (if you haven't already)
  - Create a new branch on the fork called, e.g., `fix-2`
  - Solve the issue with one or more commits
  - Push the branch to your GitHub fork
  - On GitHub, go to "Pull Requests" and open a pull request against branch `main` of the official ASPP repository
  - In the PR description write "Fixes #2" somewhere, this is going to create an automatic link to the issue, and close the issue if the PR is merged

Up next: Testing patterns

Testing error control

- Check that an exception is raised:

```
from py.test import raises
def test_raises():
    with raises(SomeException):
        do_something()
        do_something_else()
```

- For example:

```
with raises(ValueError):
    int('XYZ’)
```

passes, because

```
int('XYZ’)
ValueError: invalid literal for int() with base 10: 'XYZ'
```

Testing error control

- Use the most specific exception class, or the test may pass because of collateral damage:

```
# Test that file "None" cannot be opened.
with raises(IOError):                          => fail
   open(None, 'r')
```

as expected, but

```
                                               => pass
with raises(Exception):
   open(None, 'r’)
```