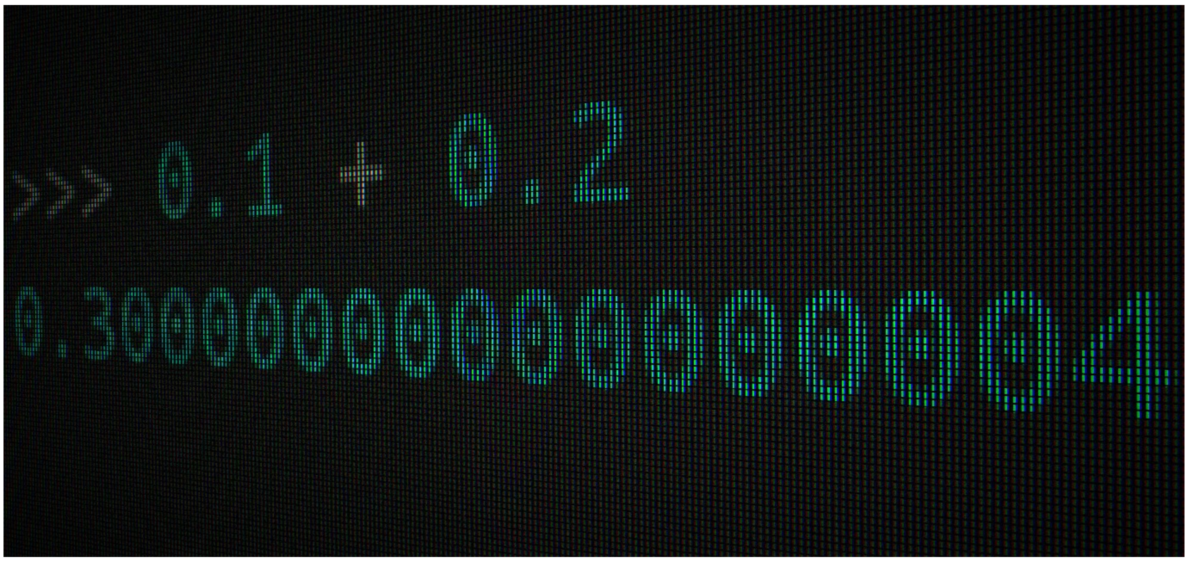
Testing and validation approaches for scientific software



Juan Luis Cano - 2018-09-25 OSCW '18 @ ESAC, Madrid

Duration

12 minutes of talk + 8 minutes of Q&A

Description

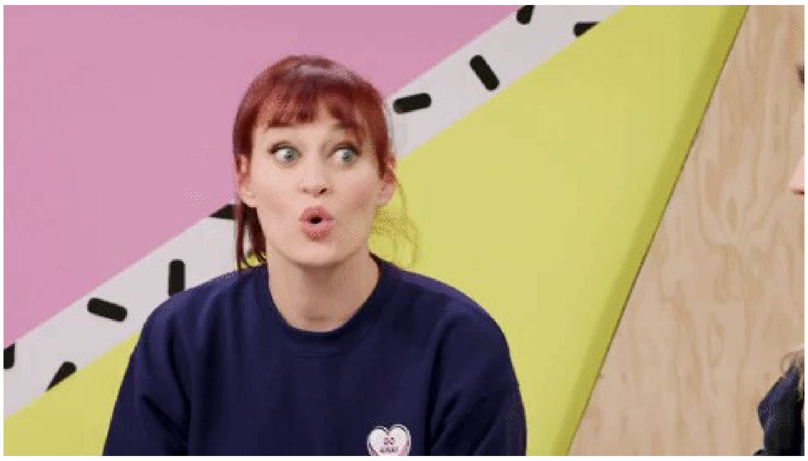
https://2018.oscw.space/event/1/contributions/13/ (https://2018.oscw.space/event/1/contributions/13/)

Nowadays, even though software has a fundamental role in scientific research, the wide majority of scientists is primarily self-taught and received no formal training in software engineering, with often leads to quality and reproducibility problems[1]. The space industry is in a similar situation, with many incident reports describing “various aspects of complacency and a discounting or misunderstanding of the risks associated with software”[2][3].

One of the most useful engineering techniques, **software testing**, is also the one that presents the biggest gap between its perceived importance and the skill level of scientists in it[4]. Testing, as well as other good practices such as version control and code reviews, not only make code more reusable but also increase the productivity of the developer[5]. However, the **special nature of scientific or algorithmic software** makes it difficult to apply commonplace testing practices, since the challenges lie in “separating software bugs from model errors and approximation error”[4].

In this talk we will discuss some testing approaches (or lack thereof) present in scientific software that fall short in helping the developers find errors or increase their productivity, and propose some other strategies based on our experience with poliastro, an open source Python library for Astrodynamics[6]. These strategies make use of **automated testing frameworks**, **help covering test cases in an exhaustive way** (hypotheses), take advantage of **analytical solutions** of the problems at hand or **public data** when available, and guarantee **self consistency** (regression testing) when there is nothing to compare against. Finally, we will analyze the limitations of these approaches and discuss possible solutions.

- [1]: Wilson, Greg, Dhavide A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven HD Haddock et al. "Best practices for scientific computing." PLoS biology 12, no. 1 (2014): e1001745.
- [2]: Leveson, Nancy G. "Role of software in spacecraft accidents." Journal of spacecraft and Rockets 41, no. 4 (2004): 564-575.



I am more like a software person, so I would like to start with a couple of questions:

- Who writes code as part of their day job?
- And who would like to find a method to write code with no errors or mistakes whatsoever??

Don't write any single line of code ever again.

Questions?

Overview

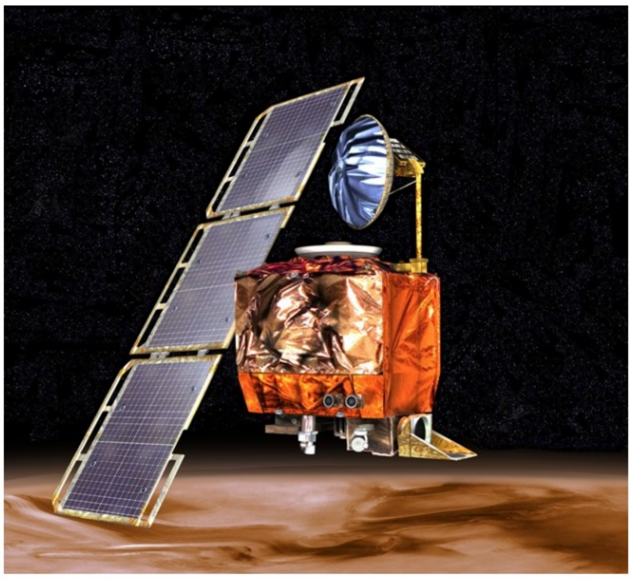
1. Introduction and motivation
2. The concept of testing and typical mistakes
3. What to validate against?
4. Python testing frameworks
5. Conclusions

And, by the way, **this talk is online!** https://github.com/poliastro/oscw2018-talk (https://github.com/poliastro/oscw2018-talk) (go, start it ⭐)

A not so long time ago...

https://www.wired.com/2010/11/1110mars-climate-observer-report/
(https://www.wired.com/2010/11/1110mars-climate-observer-report/)

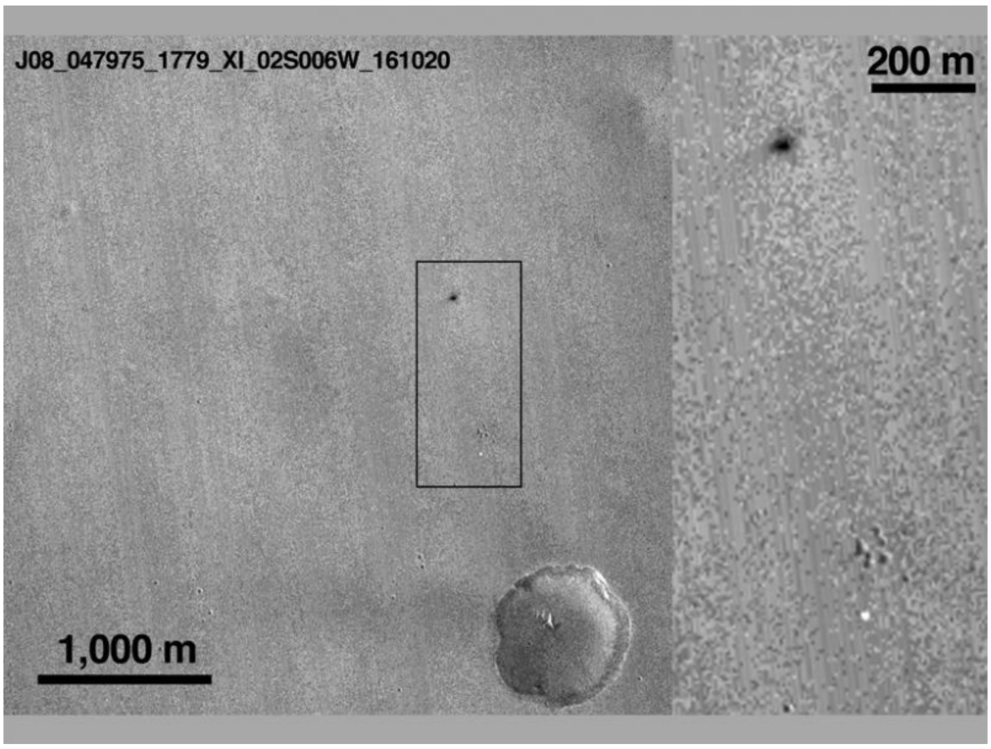
NOV. 10, 1999: METRIC MATH MISTAKE MUFFED MARS METEOROLOGY MISSION



1999: A disaster investigation board reports that NASA's Mars Climate Orbiter burned up in the Martian atmosphere because engineers failed to convert units from English to metric.

Only a few years ago...

http://www.esa.int/Our_Activities/Space_Science/ExoMars/Schiaparelli_landing_investigation_completed (http://www.esa.int/Our_Activities/Space_Science/ExoMars/Schiaparelli_landing_investigation_completed)



> Recommendation 05 – Robust and reliable sanity checks shall be implemented in the on-board S/W to increase the robustness of the design, which could be, but not limited to :
>
> - Check on attitude
> - Check on altitude sign **(altitude cannot be negative)**.
> - Check on vertical acceleration during terminal descent and landing **(cannot be higher than gravity)**.
> - Check altitude magnitude change **(it cannot change from 3.7 Km to a negative value in one second)**

(emphasis mine)

Testing

> Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test.
>
> —https://en.wikipedia.org/wiki/Software_testing (https://en.wikipedia.org/wiki/Software_testing)

Assumption #0: You have *some* tests

Examples:

- End-to-end (E2E) simulation that compares the result of today and yesterday
- A checklist that must be followed before important milestones
- An Excel spreadsheet where a person marks whether some functionality is present or not

Typical mistake #1: Not having *automated* tests

> If you use software that lacks automated tests, you are the tests.
> — Jenny Bryan (@JennyBryan) September 22, 2018 (https://twitter.com/JennyBryan/status/1043307291909316609?ref_src=twsrc%5Etfw)

- People fail while running the tests
- Confronting a coworker work can be tough
- People tend to oversee certain issues on Friday afternoons

Typical mistake #2: Not having *unit* tests

- Software is made up of small pieces (right???) so we should test them individually (i.e. and E2E simulation is not a enough)
- If a small piece of software is not testable, it's almost certainly a sign of bad design
- Code that has side effects (writes to disk, trigger the fire alarm) is inherently difficult to test, so they should be isolated as much as possible
- Keyword: **refactoring**, learn it!

Typical mistake #3: Shooting yourself in the foot

i.e. writing a passing test that is, in fact, hiding a problem.

> Example task: "Write the sinc function"

$$ \operatorname{sinc}(x) = \frac{\sin{x}}{x} $$

In [1]:
```
import numpy as np

def sinc(x):
    return np.sin(x) / x
```

In [2]:
```python
import pytest

def test_sinc():
    assert sinc(1.0) == np.sin(1.0) / 1.0
```

In [4]:
```python
# https://jakevdp.github.io/blog/2017/12/05/installing-python-packages
-from-jupyter/
import sys
!{sys.executable} -m pytest test_sinc1.py
```

```
============================= test session starts ===============
===============
platform linux -- Python 3.7.0, pytest-3.8.0, py-1.5.4, pluggy-0.
7.1
hypothesis profile 'default' -> database=DirectoryBasedExampleDat
abase('/home/juanlu/Development/poliastro/talks/oscw2018/.hypothe
sis/examples')
rootdir: /home/juanlu/Development/poliastro/talks/oscw2018, inifi
le:
plugins: cov-2.5.1, hypothesis-3.73.0
collected 1 item

test_sinc1.py .
[100%]

=========================== 1 passed in 0.09 seconds ============
===============
```

Potential mistakes:

- Copy-paste: it's very tempting to copy the function to the test, isn't it? Looks like a great way to **copy-paste typos and mistakes as well**!
- Corner cases (see next point)
- Bogus floating point comparisons!

In [5]:
```python
0.1 + 0.2 == 0.3
```

Out[5]:
```
False
```

In [6]:
```python
0.2 + 0.3 == 0.5
```

Out[6]:
```
True
```

Typical mistake #4: Not covering corner cases

In [7]:
```python
import pytest

# https://docs.pytest.org/en/stable/parametrize.html
@pytest.mark.parametrize("x", [-2, 1, 1, 2])
def test_sinc(x):
    assert sinc(x) == np.sin(x) / x
```

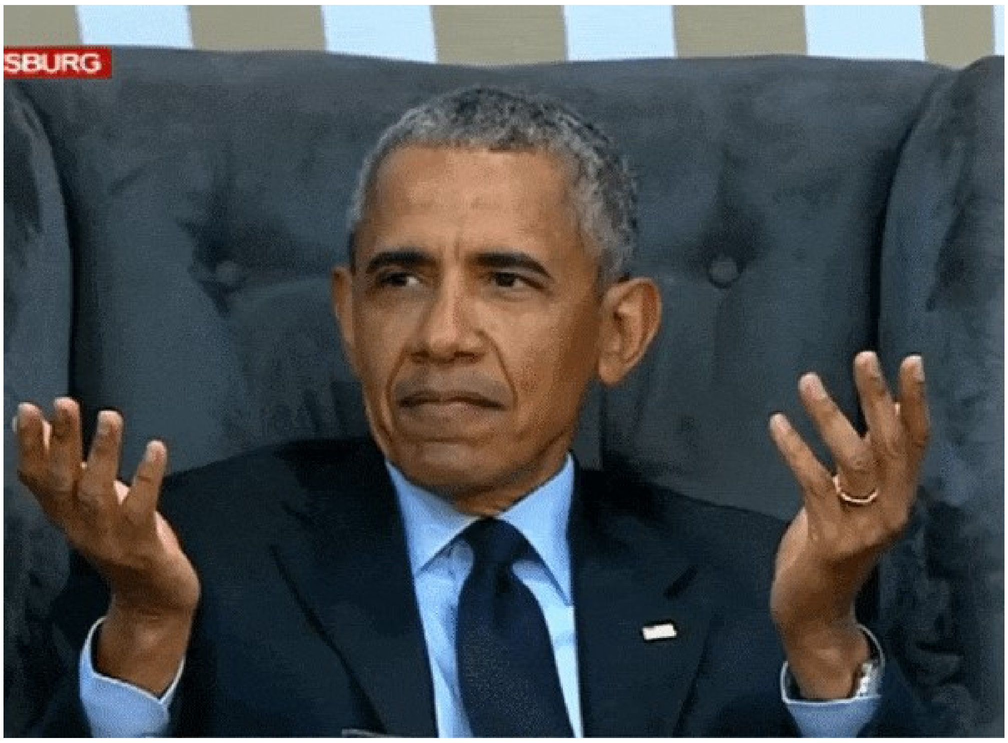
In [9]:
```
!{sys.executable} -m pytest test_sinc2.py
```

```
============================== test session starts ===============
================
platform linux -- Python 3.7.0, pytest-3.8.0, py-1.5.4, pluggy-0.
7.1
hypothesis profile 'default' -> database=DirectoryBasedExampleDat
abase('/home/juanlu/Development/poliastro/talks/oscw2018/.hypothe
sis/examples')
rootdir: /home/juanlu/Development/poliastro/talks/oscw2018, inifi
le:
plugins: cov-2.5.1, hypothesis-3.73.0
collected 4 items

test_sinc2.py ....
[100%]

============================ 4 passed in 0.09 seconds ============
================
```

In [10]:
```
sinc(0) == np.sin(0) / 0
```

```
/home/juanlu/.miniconda36/envs/poliastro37/lib/python3.7/site-pac
kages/ipykernel_launcher.py:4: RuntimeWarning: invalid value enco
untered in double_scalars
  after removing the cwd from sys.path.
/home/juanlu/.miniconda36/envs/poliastro37/lib/python3.7/site-pac
kages/ipykernel_launcher.py:1: RuntimeWarning: invalid value enco
untered in double_scalars
  """Entry point for launching an IPython kernel.
```

Out[10]:
```
False
```



In [11]:
```
from hypothesis import given
import hypothesis.strategies as st

@given(st.floats())
def test_sinc(x):
    assert sinc(x) == np.sin(x) / x
```

In [14]:
```
def sinc(x):
    return np.sin(x) / x if x != 0.0 else 1.0
```

In [15]:
```
@given(st.floats())
def test_sinc(x):
    assert sinc(x) == np.sin(x) / x  # And now what?
```

We need validation data!

What to validate against?

A better way:

- Compare against some authoritative source: **external data or software**
- Do floating point comparisons right and **use tolerances**

In [16]:
```
def test_convert_from_rv_to_coe():
    # Data from Vallado, example 2.6
    attractor = Earth
    p = 11067.790 * u.km
    ecc = 0.83285 * u.one
    inc = 87.87 * u.deg
    raan = 227.89 * u.deg
    argp = 53.38 * u.deg
    nu = 92.335 * u.deg
    expected_r = [6525.344, 6861.535, 6449.125] * u.km
    expected_v = [4.902276, 5.533124, -1.975709] * u.km / u.s

    r, v = ClassicalState(attractor, p, ecc, inc, raan, argp, nu).rv()

    assert_quantity_allclose(r, expected_r, rtol=1e-5)
    assert_quantity_allclose(v, expected_v, rtol=1e-5)
```

What's the perfect way?

- How much precision do you ask for? Should you carry a mathematical analysis?
- What if your results don't match? Sometimes, book or paper authors respond to your comments... And sometimes don't
- The changes in precision are a result of bad data, or worse algorithms?
- How do you even track *improvements*?

External data (short summary)

- Nobody cares

- Those who care, don't share it

- Those who share, do it with 1 decimal place (true story)

- Those who share with 16 decimal places, don't describe how it was obtained (i.e. release the source)

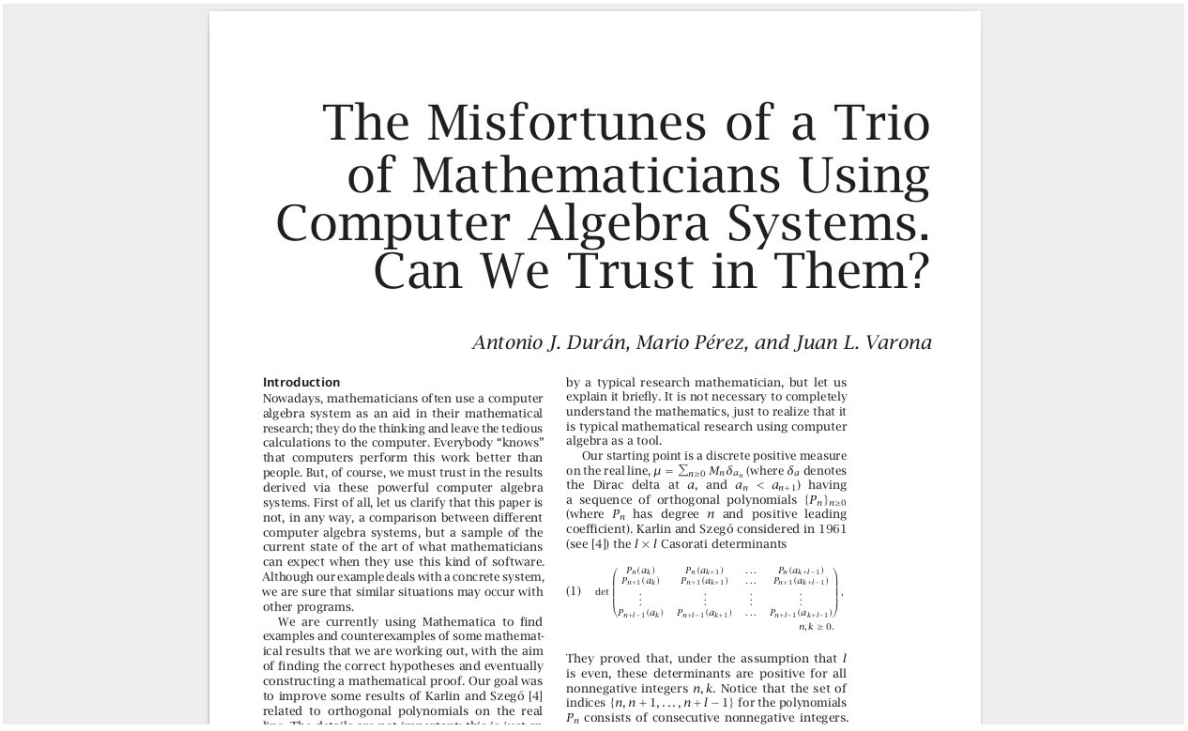
- Those who release the source, make it impossible to compile

External software

- Sometimes commercial
- It is often difficult to reproduce the exact setting and algorithms, most of the times because your commercial software is much more complex

...but is it validated itself?

[https://www.ams.org/notices/201410/rnoti-p1249.pdf (https://www.ams.org/notices/201410/rnoti-p1249.pdf)](https://www.ams.org/notices/201410/rnoti-p1249.pdf)



The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?

Antonio J. Durán, Mario Pérez, and Juan L. Varona

Introduction

Nowadays, mathematicians often use a computer algebra system as an aid in their mathematical research; they do the thinking and leave the tedious calculations to the computer. Everybody "knows" that computers perform this work better than people. But, of course, we must trust in the results derived via these powerful computer algebra systems. First of all, let us clarify that this paper is not, in any way, a comparison between different computer algebra systems, but a sample of the current state of the art of what mathematicians can expect when they use this kind of software. Although our example deals with a concrete system, we are sure that similar situations may occur with other programs.

We are currently using Mathematica to find examples and counterexamples of some mathematical results that we are working out, with the aim of finding the correct hypotheses and eventually constructing a mathematical proof. Our goal was to improve some results of Karlin and Szegő [4] related to orthogonal polynomials on the real

by a typical research mathematician, but let us explain it briefly. It is not necessary to completely understand the mathematics, just to realize that it is typical mathematical research using computer algebra as a tool.

Our starting point is a discrete positive measure on the real line, $\mu = \sum_{n\geq 0} M_n \delta_{a_n}$ (where δ_a denotes the Dirac delta at a, and $a_n < a_{n+1}$) having a sequence of orthogonal polynomials $\{P_n\}_{n\geq 0}$ (where P_n has degree n and positive leading coefficient). Karlin and Szegő considered in 1961 (see [4]) the $l \times l$ Casorati determinants

$$(1) \quad \det \begin{pmatrix} P_n(a_k) & P_n(a_{k+1}) & \dots & P_n(a_{k+l-1}) \\ P_{n+1}(a_k) & P_{n+1}(a_{k+1}) & \dots & P_{n+1}(a_{k+l-1}) \\ \vdots & \vdots & \vdots & \vdots \\ P_{n+l-1}(a_k) & P_{n+l-1}(a_{k+1}) & \dots & P_{n+l-1}(a_{k+l-1}) \end{pmatrix}, \quad n, k \geq 0.$$

They proved that, under the assumption that l is even, these determinants are positive for all nonnegative integers n, k. Notice that the set of indices $\{n, n+1, \dots, n+l-1\}$ for the polynomials P_n consists of consecutive nonnegative integers.

Conclusions

- Writing good, comprehensive tests is not trivial
- But we need to!
- There are tools at hand that help
- Automate **everything**
- Validate what you use, it might bite you

Per Python ad astra! 🚀

poliastro

Astrodynamics in Python

- Slides: https://github.com/poliastro/oscw2018-talk (https://github.com/poliastro/oscw2018-talk)
- poliastro chat: https://riot.im/app/#/room/#poliastro:matrix.org (https://riot.im/app/#/room/#poliastro:matrix.org)
- Twitter: https://twitter.com/poliastro_py (https://twitter.com/poliastro_py)