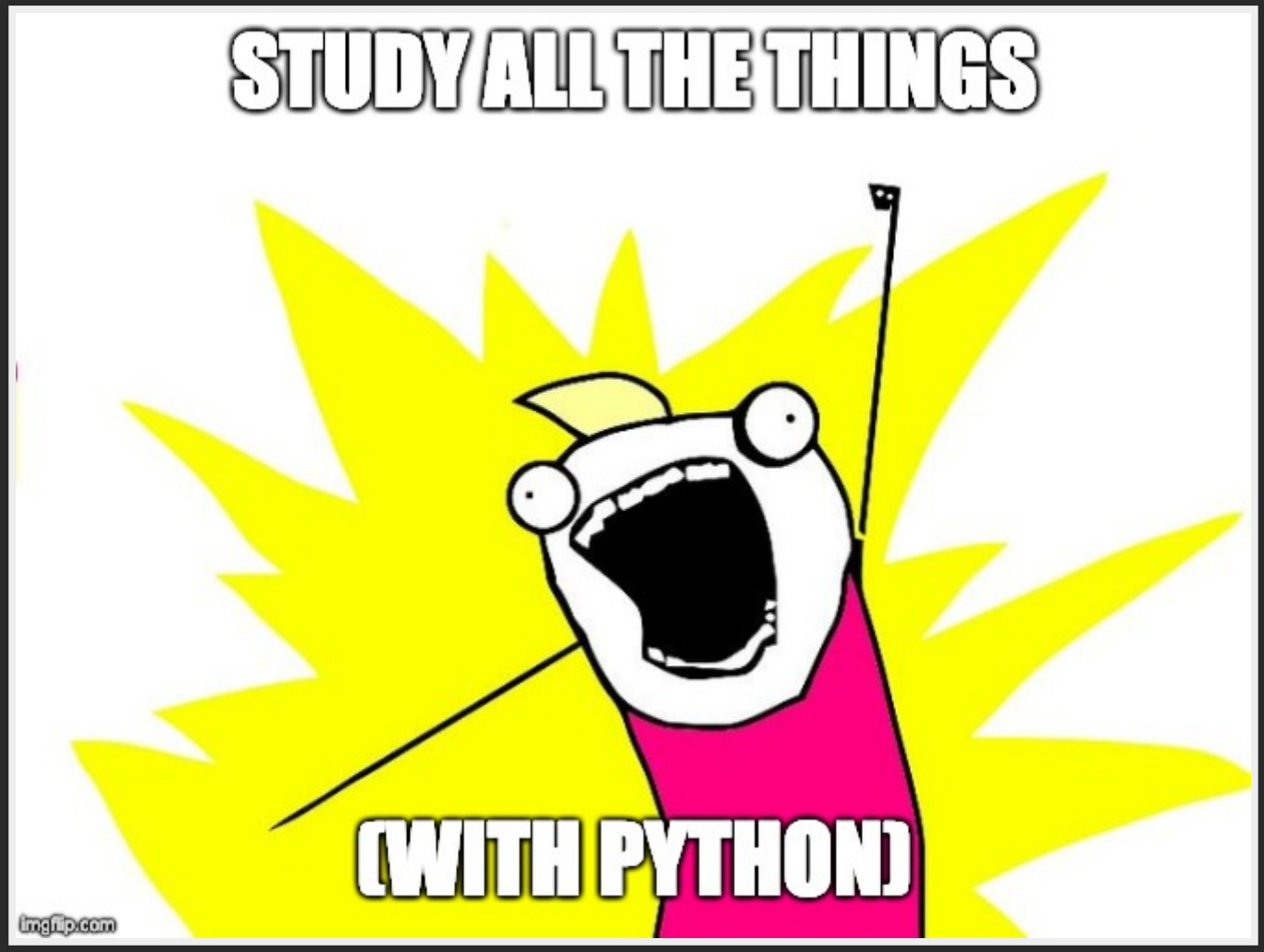
ABSTRACTION FOR STUDENTS OF ALL THE THINGS

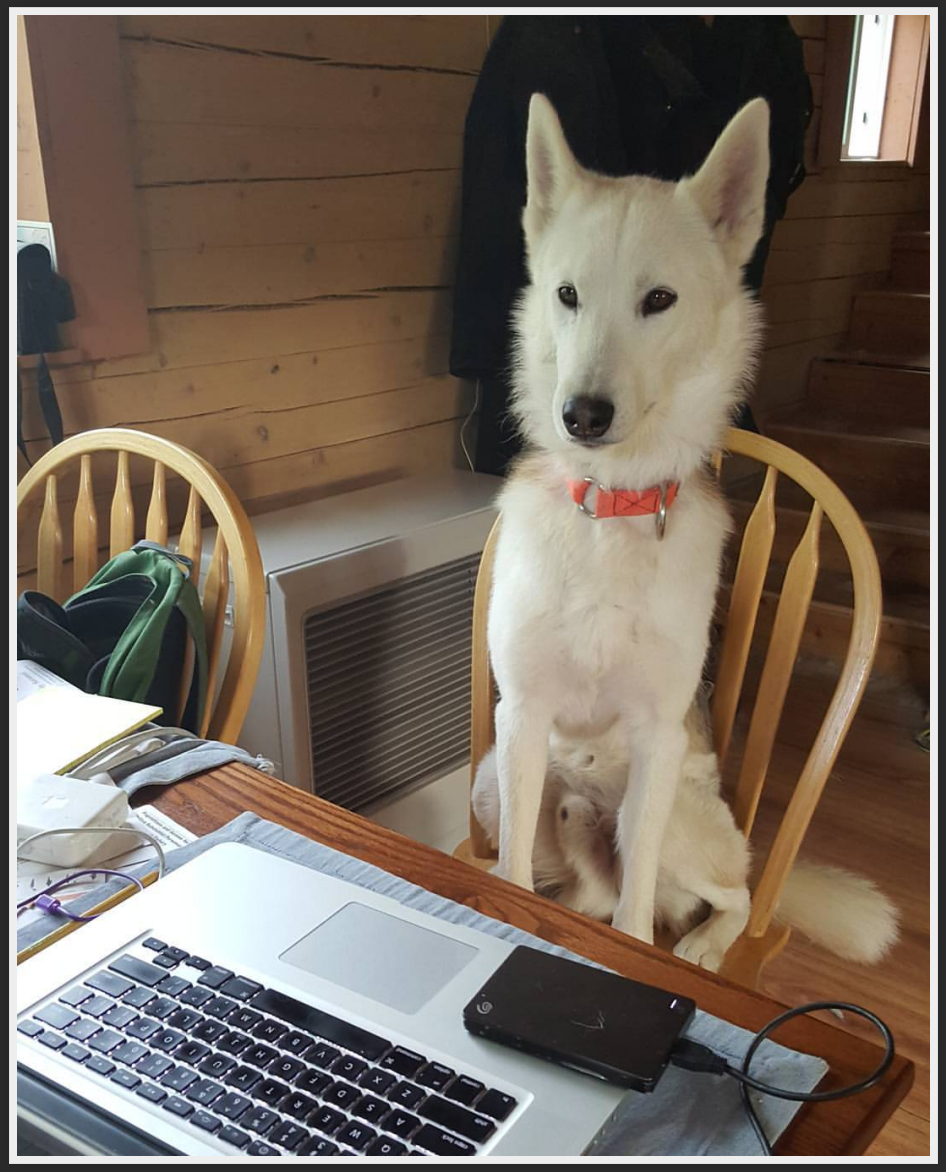
PyCascades 2019

Chris Waigl

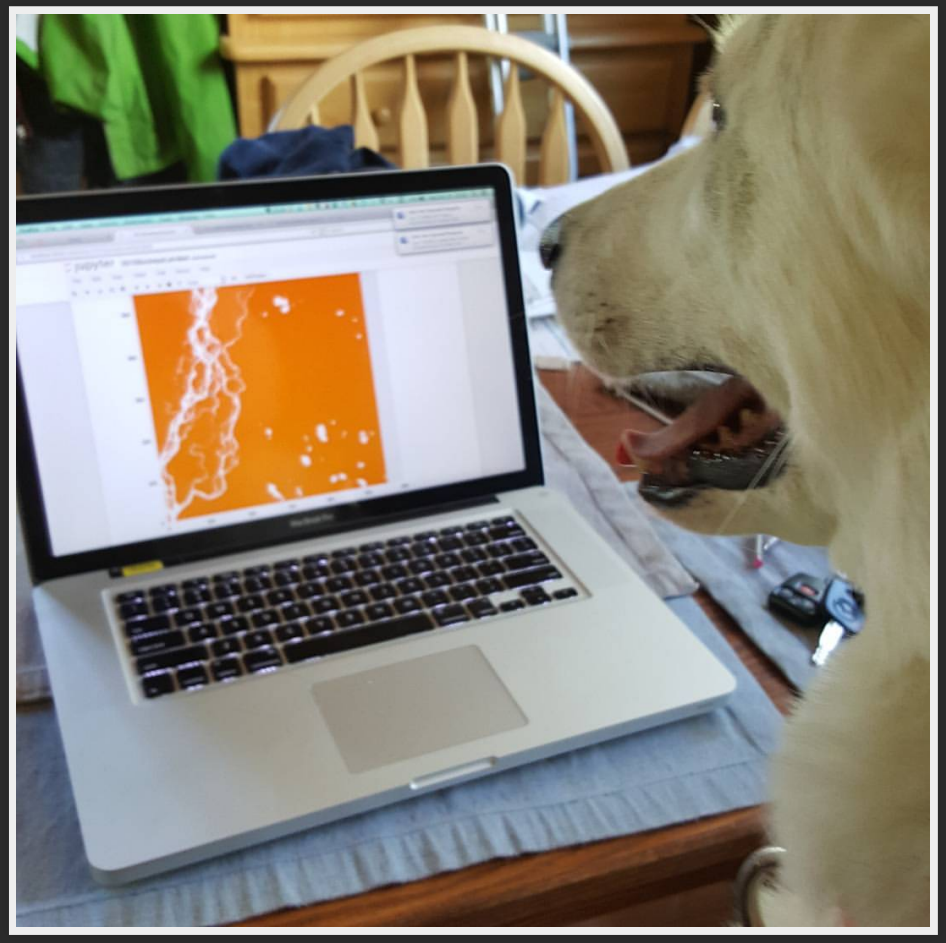


See Hyperbole and a Half

I'm an earth scientist.



Dirty secret: Scientists often write mediocre or outright bad code.



Alice Harpole: https://www.software.ac.uk/blog/2017-09-12-how-write-code-scientist

How to write code like a scientist

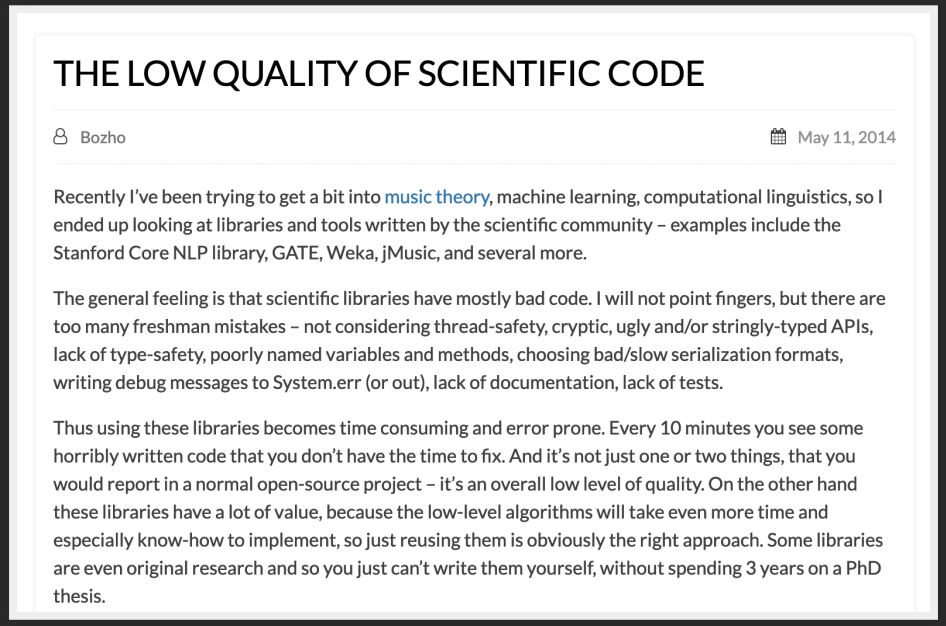
Posted by s.aragon on 12 September 2017 - 9:37am

By Alice Harpole, University of Southampton

Coding is often seen as a tool to do science, rather than an intrinsic part of the scientific process. This often results in scientific code that is written in a rather unscientific way. In my experience as a PhD student, I've regularly read papers describing exciting new codes, only to find that there are number of issues preventing me from looking at or using the code itself. The code is often not open source, which means I can't download or use it. Code commonly has next to no documentation, so even if I can download it, it's very difficult to work out how it runs. There can be questionable approaches to testing with an overreliance on replicating "standard" results, but no unit tests exist to demonstrate that the individual parts of the code work as they should. This

"I've regularly read papers describing exciting new codes, only to find that there are number of issues preventing me from looking at or using the code itself. The code is often not open source, which means I can't download or use it. Code commonly has next to no documentation, so even if I can download it, it's very difficult to work out how it runs. There can be questionable approaches to testing with an overreliance on replicating "standard" results, but no unit tests exist to demonstrate that the individual parts of the code work as they should."

Bozhidar Bozhanov: https://techblog.bozho.net/the-astonishingly-low-quality-of-scientific-code/



THE LOW QUALITY OF SCIENTIFIC CODE

Bozho — May 11, 2014

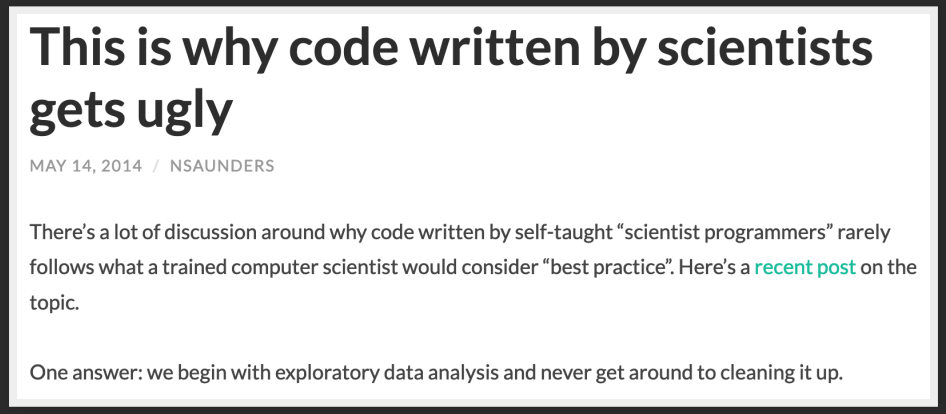
Recently I've been trying to get a bit into music theory, machine learning, computational linguistics, so I ended up looking at libraries and tools written by the scientific community – examples include the Stanford Core NLP library, GATE, Weka, jMusic, and several more.

The general feeling is that scientific libraries have mostly bad code. I will not point fingers, but there are too many freshman mistakes – not considering thread-safety, cryptic, ugly and/or stringly-typed APIs, lack of type-safety, poorly named variables and methods, choosing bad/slow serialization formats, writing debug messages to System.err (or out), lack of documentation, lack of tests.

Thus using these libraries becomes time consuming and error prone. Every 10 minutes you see some horribly written code that you don't have the time to fix. And it's not just one or two things, that you would report in a normal open-source project – it's an overall low level of quality. On the other hand these libraries have a lot of value, because the low-level algorithms will take even more time and especially know-how to implement, so just reusing them is obviously the right approach. Some libraries are even original research and so you just can't write them yourself, without spending 3 years on a PhD thesis.

"Scientists in general can't write good code. They write code simply to achieve their immediate goal, and then either throw it away, or keep using it for themselves. They [...] don't seem to be concerned with code quality, code coverage, API design. Not to mention scientific infrastructure, deployment on multiple servers, managing environment. These things are rarely done properly in the scientific community. "

Neil Saunders:
https://nsaunders.wordpress.com/2014/05/14/this-is-why-code-written-by-scientists-gets-ugly/



This is why code written by scientists gets ugly

MAY 14, 2014 / NSAUNDERS

There's a lot of discussion around why code written by self-taught "scientist programmers" rarely follows what a trained computer scientist would consider "best practice". Here's a recent post on the topic.

One answer: we begin with exploratory data analysis and never get around to cleaning it up.

"One answer: we begin with exploratory data analysis and never get around to cleaning it up. "

REMEDIES?

Teach scientists code organization, version control, software engineering practices

Hire dedicated scientific programmers

"It's all good, scientists need to make a mess!" NOT!

... ok, but ...

WHAT ABOUT THE QUALITY OF THE CODE ITSELF?

We can *and should* also work on writing better code.

Thinking about code through the lens of abstraction is going to help us.

ABSTRACTION AND MENTAL CHUNKING

(abstract != concrete)

ABSTRACTION IN THE PYTHON DOCUMENTATION

The Python Language Reference

3.1. Objects, values and types

Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects. (In a sense, and in conformance to Von Neumann's model of a "stored program computer," code is also represented by objects.)

Module documentation of `urllib` and `ssl`

class `urllib.request.` **`Request`**(*url*, *data=None*, *headers={}*, *origin_req_host=None*, *unverifiable=False*, *method=None*)

This class is an abstraction of a URL request.

However, since the SSL (and TLS) protocol has its own framing atop of TCP, the SSL sockets abstraction can, in certain respects, diverge from the specification of normal, OS-level sockets. See especially the notes on non-blocking sockets.

The Programming FAQ warns of "excessive abstraction" (in the "performance" section).

- Abstractions tend to create indirections and force the interpreter to work more. If the levels of indirection outweigh the amount of useful work done, your program will be slower. You should avoid excessive abstraction, especially under the form of tiny functions or methods (which are also often detrimental to readability).

(Note the notion of *levels* of abstraction.)

Basic abstraction-related example in student code.

```
with open("data_file.json", "r") as read_file:
    data = json.load(read_file)
```

```
json_string = """
{
    "dataset": {
        "temp": 28.4,
        "species": "Betula",
        "datestamp": "2019-01-03 14:55:22"
    }
}
"""
data = json.loads(json_string)
```

```
>>> data['dataset']['temp']
28.4
>>> data['dataset']['datestamp']
'2019-01-03 14:55:22'
```

```
>>> type(data['dataset']['datestamp'])
```

- Not a datestamp, or a datetime, but a string.

FIRST ENCOUNTERS WITH ABSTRACTION IN PYTHON

DUCK TYPING IS ABOUT ABSTRACTION

Let's quote the Python Language Reference again:

> *An object's type determines the operations that the object supports*

For example, iterating through a list.

Step 1: A beginner who writes this...

```
mylist = [1, 5, 8, 2, 9]
for ii in range(len(mylist)):
    do_something(mylist[ii])
```

... thinks of a list as some sort of array with an index that can be incremented.

Step 2: When we teach her to write code like this ...

```
mylist = [1, 5, 8, 2, 9]
for element in mylist:
    do_something(element)
```

... we teach a different mental concept of a list: a list is composed of elements that can be accessed sequentially.

Step 3: Then we generalize: the concept is really about iterables:

```
myiterable = ... # could be list, set, tuple, string, dictionary
for element in myiterable:
    do_something(element)
```

Note: The consistency of Python becomes evident. Compare with MATLAB:

```
v = [1 5 8 17]
for element = v
    do_something(element)
end
```

Another example in Python: file-like objects.

Using the `io` module (`StringIO` and `BytesIO` we can treat strings as streams the same way as we do files: We can open and close them, read from them with buffer, write to them....

(Sockets and serial interfaces work very much the same way.)

WRITING YOUR OWN ABSTRACTIONS

... BY WRITING FUNCTIONS:

- How do you segment the processing flow into functions? (... and modules, packages...)
- This is an expression of how you conceptualize your task
- You can: Re-use. Pass around data. Hide implementation choices behind interfaces.

Abstraction for problem-solving

... BY CHOOSING YOUR DATA REPRESENTATION:

- What comes first, the data or the functions that handle the data?
- The abstractions that represent data and the abstractions that represent programming steps will be developed iteratively, hand-in-hand.
- Data representation means to decide what the objects *are* that your code is manipulating

Abstraction for reasoning about the overall shape of your problem

TYING IT TOGETHER

- The transition to object-oriented design becomes natural
- Many scientific Python users never write their own objects. But it becomes easy once we have clarified what our objects are, how they behave, and what operations are performed on them.

(And we have reached the point where we can talk about abstraction and encapsulation.)

ABSTRACTION PITFALLS

ABSTRACTIONS LEAK.
PYTHON ABSTRACTIONS LEAK QUITE A BIT.

- Error messages and exceptions may refer to underlying code that is at a deeper abstraction level
- Objects are not watertight: `myobject.__dict__` allows access to attributes under the hood.

SOME PYTHON GOTCHAS ARE RELATED TO ABSTRACTION

The mutable default function argument gotcha.

REMEMBER THE WARNING ABOUT PERFORMANCE FROM THE PROGRAMMING FAQ?

- "Abstraction, simplicity, performance: choose two"
- Scientific Python libraries to the rescue: They offer a wide choice of abstractions that are optimised for performance or expressiveness.

https://github.com/chryss/abstraction-for-students-of-all-the-things

chryss @chrys

chris@threeowlsdata.com