

National Synchrotron Light Source II



Designing Open Source Code Ecosystems

Daniel Allan

Data Engineering Group Lead

Data Science and Systems Integration Program, NSLS-II

RSoXS Code Camp, NIST, August 2023

Who Am I

- Ph.D. in Experimental Condensed Matter Physics (JHU)
- 8 years in a *de facto* Research Software Engineering role at NSLS-II
- Started the *Bluesky* project, a green-field (blue sky...) open-source Python approach to data acquisition and data access experimental science, with collaborators Thomas Caswell and Ken Lauer
- *Bluesky* has grown within NSLS-II and around the world

99-80
September 8, 1999

For more information, contact:
Diane Greenberg, (631)344-2347, greenb@bnl.gov,
or Mona S. Rowe, (631) 344-5056, mrowe@bnl.gov

Brookhaven Lab to Hold "Open Source/Open Science" Conference, Oct. 2

UPTON, NY - The U.S. Department of Energy's Brookhaven National Laboratory will host a conference titled "Open Source/Open Science" on Saturday, October 2, from 8 a.m. to 5 p.m. in the Laboratory's Berkner Hall. The public is invited to the conference, but pre-registration is required. The fee for the conference is \$25, which includes lunch.

Open Source software is free software, typically released over the Internet so that a broad audience can evaluate and test it. The conference will bring together scientists and developers to exchange ideas about the use of Open Source software in scientific research at Brookhaven, as well as at other laboratories and universities. Talks by invited speakers, posters, demonstrations and vendor exhibits will be featured at the conference. Also offered will be tours of some of Brookhaven Lab's facilities in which Open Source software is used.

Among the talks and speakers featured at the conference will be:

- "What is Open Source?" Bruce Perens, the principal author of the Open Source definition
- "The Open Science Project," by Dan Gezelter from openscience.org
- "Open Source Computing for BNL's Relativistic Heavy Ion Collider," Tom Throwe, Brookhaven Lab
- "Open Visualization Data Explorer," Bill Horn, IBM
- "Open GL and GLX: High Performance 3D Graphics for Linux," Jon Leech, SGI
- "ACEDB: An Open Source Object-Oriented Database," Lincoln Stein, Cold Spring Harbor Laboratory
- "Open Source in Medical Imaging," Bill Rooney, Brookhaven Lab
- "Using Beowulf for Macromolecular Crystallography at the National Synchrotron Light Source," Malcolm Capel, Brookhaven Lab

Also, a panel of experts will discuss the issues that must be overcome by federal facilities in order to use and contribute to Open Source technologies.

To register for the conference, go to the Web site <http://openscience.bnl.gov>, and follow the registration instructions. The registration deadline is September 26. For more information, call (516)344-3582 or (516)344-5682.

Brookhaven National Laboratory is located on William Floyd Parkway (County Road 46), one-and-a-half miles north of Exit 68 on the Long Island Expressway.

The U.S. Department of Energy's Brookhaven National Laboratory creates and operates major facilities available to university, industrial and government personnel for basic and applied research in the physical, biomedical and environmental sciences, and in selected energy technologies. The Laboratory is operated by Brookhaven Science Associates, a not-for-profit research management company, under contract with the U.S. Department of Energy.

* * *

experiment orchestration and data acquisition <https://blueskyproject.io/bluesky/>

Edit

python Manage topics

3,695 commits

9 branches

46 releases

24 contributors

View license

Branch: master

New pull request

Create new file

Upload files

Find File

Clone or download

awalter-bnl Merge pull request #1236 from mrakitin/fix-docs-persistentdict Latest commit 32cc867 7 days ago

.github DOC: removed checkbox section 3 years ago

bluesky Merge pull request #1232 from dmgav/thread-problem-test 14 days ago

doc DOC: minor correction 8 days ago

What do we mean "open source"?

1. Publicly visible source code
2. Licensed for reuse with an OSI-approved license
3. Accepting contributions
4. Open development
5. Open decision making
6. Multi-institution engagement
7. Retirement



from Matt Rocklin's post
<https://www.coiled.io/blog/stages-of-openness>

Some Technical Goals of Bluesky

- Be generic across science domains.
- Be unopinionated data *formats*; focus on data *structures*.
- Handle asynchronous data streams.
- Support multi-modal: simultaneous, cross-instrument, cross-facility experiments.
- Support streaming.



Some Sociological Goals of Bluesky

- Overcome "not-invented-here"-ism.
- Make *co-developed but separately useful* components with well-defined boundaries which can be adopted piecemeal by other groups and facilities.
- Enable code to be reused in ways unforeseen by the original authors.



Work openly from the start

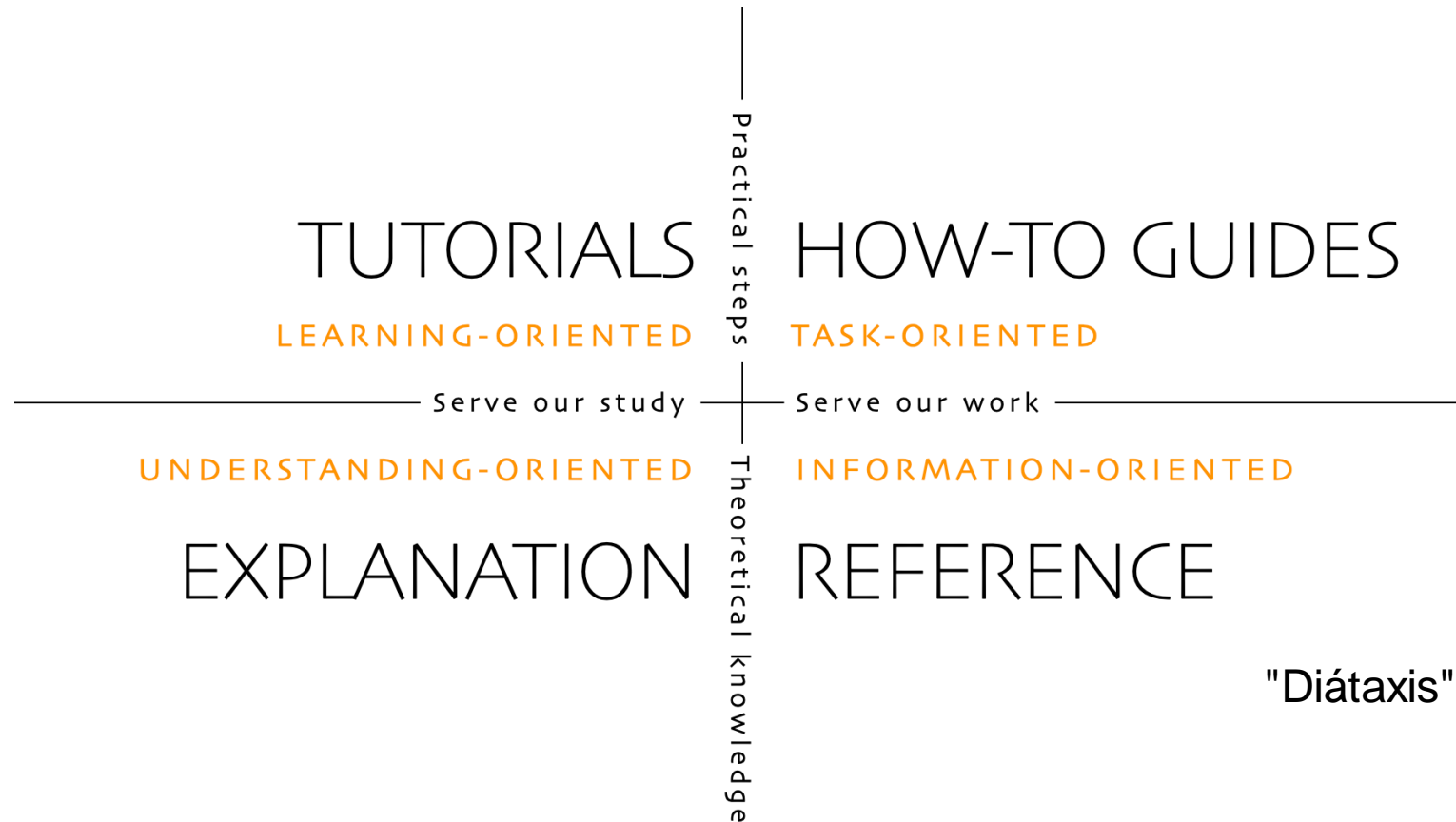
- Discover potential collaborators early, before it is costly to consolidate.
- Different perspectives can identify where code may need to be flexible to support future use cases.
- Having more than one person understanding every part of the code prevents systematic risks for the project and keeps you from being tied to that code.

Automated tests are essential



- They enable people to try new ideas with confidence.
- Ensure that we don't accidentally break our ability to recreate important results.
- Ensure that my "improvement" won't accidentally break your research code by protecting it with tests that verify key results.
- Continuous Integration services ensure the tests always get run on every proposed change.

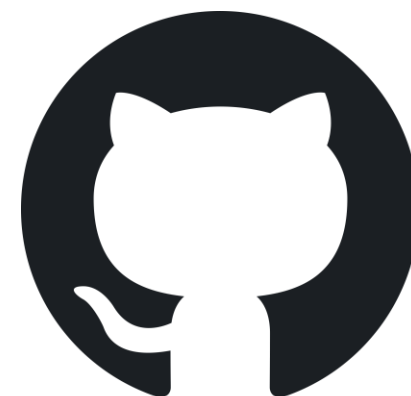
Good, current documentation can convince people it is easier to learn to use your project than write their own



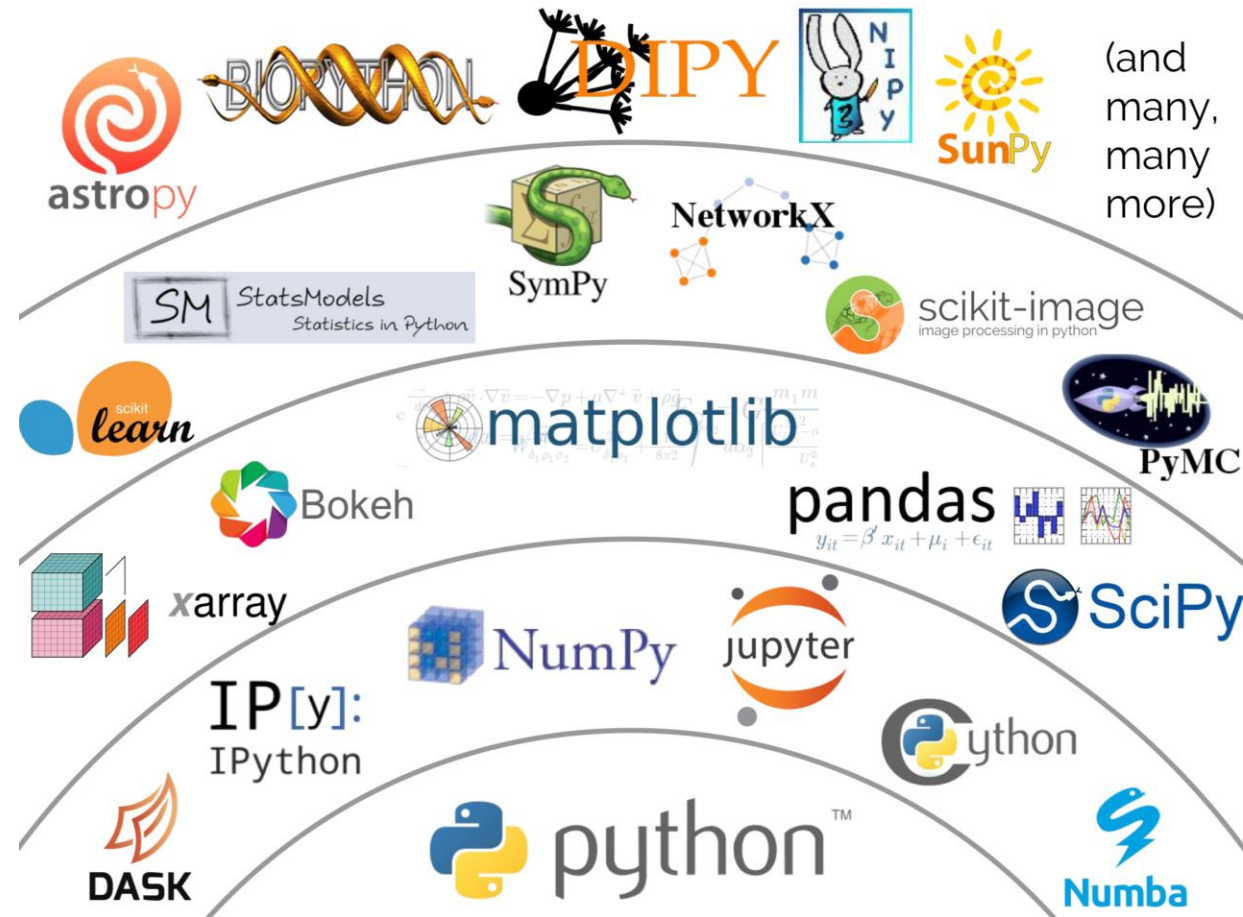
"Diátaxis" <https://diataxis.fr>

Minimum Viable Governance

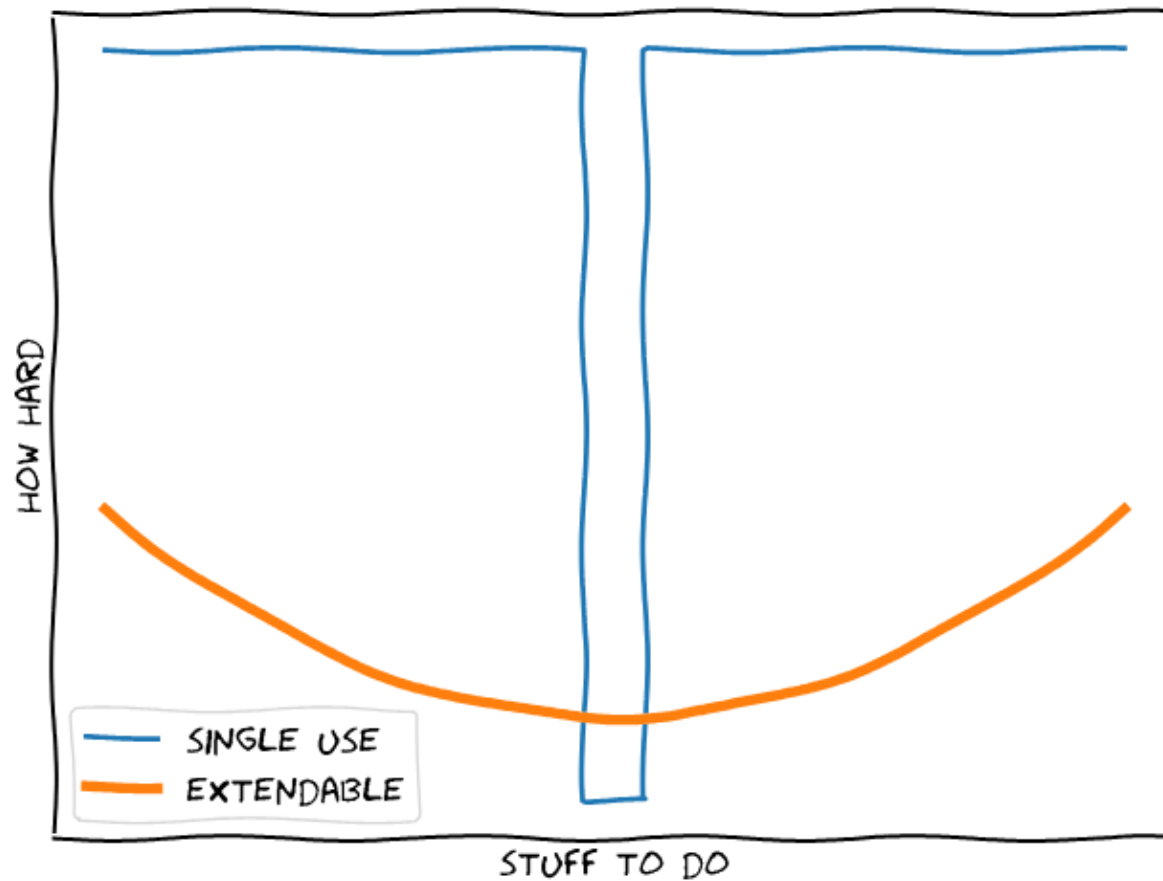
- Maintainers: per repo, make day-to-day decisions and set processes as appropriate to the repo
- Technical Steering Committee: arbitrate when maintainers cannot reach rough consensus
- Project Advisory Board: management-level stakeholders, oversee big-picture priorities



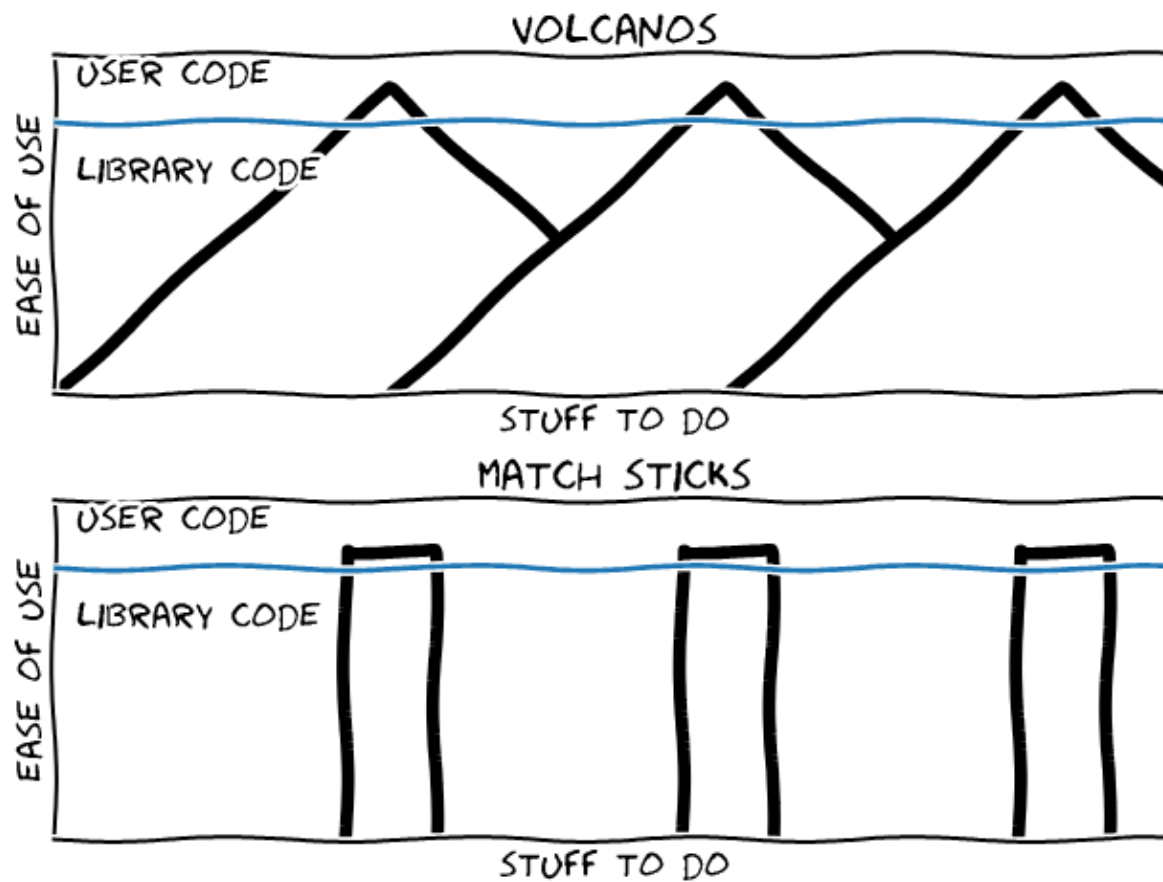
Scientific Python is a *layered* ecosystem



Aim for layered, extensible code



Aim for layered, extensible code



Embrace Protocols

- *Protocols* enable interoperable tools without explicit coordination
- This has been the key to the success of Scientific Python
- And mini-ecosystems around it, like Bluesky

Example: Python iteration protocol

```
for i in range(10):
```

```
    ...
```

Example: Python iteration protocol

```
class Thing:  
    def __iter__(self):  
        ...
```

```
for x in Thing():  
    ...
```

Example: numpy `__array__` protocol

```
import pandas
import numpy

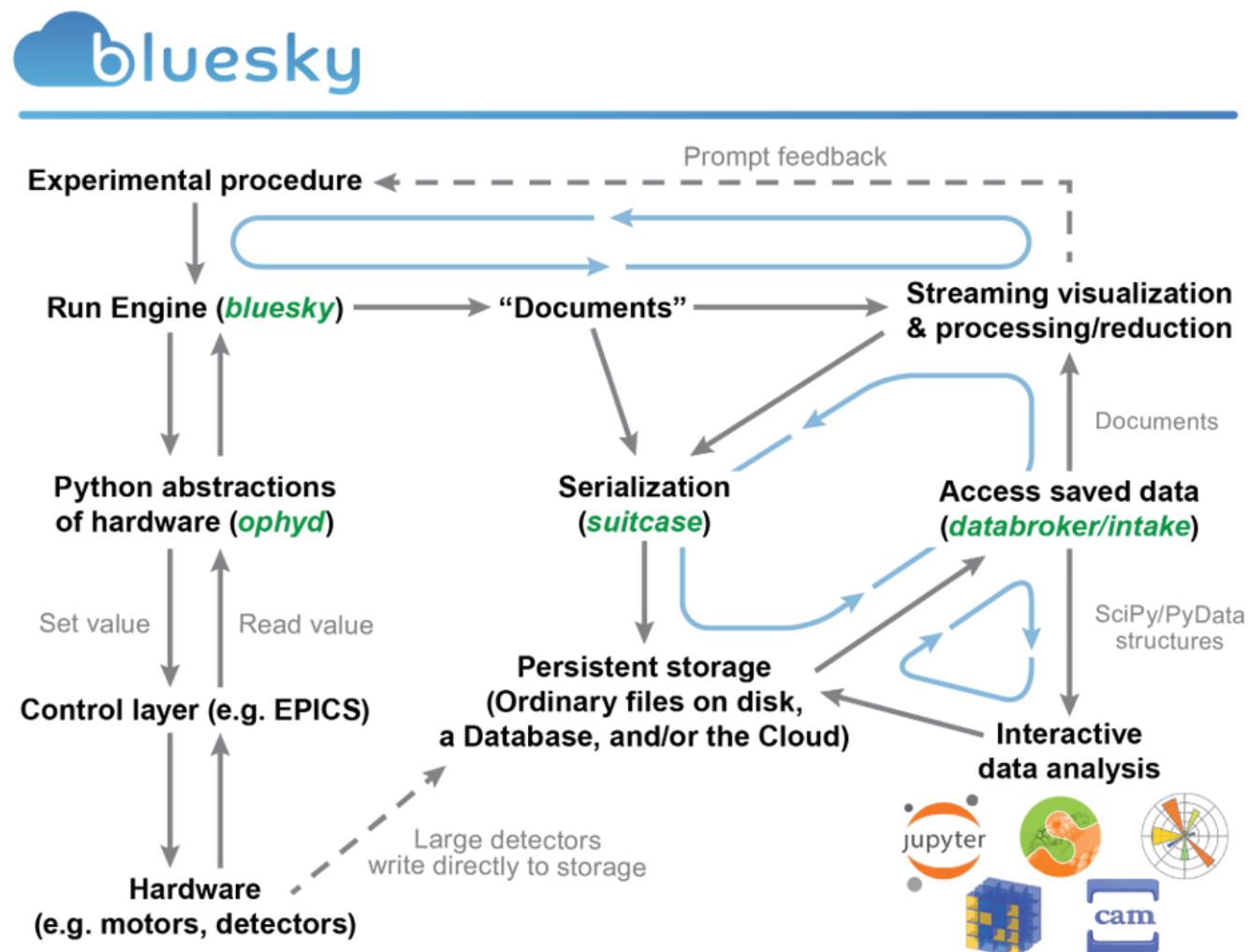
df = pandas.DataFrame({'intensity': [1,1,2,3]})
numpy.sum(df)    # How does this work?
```

Example: scikit-learn Pipelines

- Estimators implement `estimator.fit(data, [targets])`
- Predictor implement `predictor.predict(data)`
- Transformers implement `transformer.transform(data)`
- Models implement `model.score(data)`

Example: Protocols in Bluesky

- Device protocol
- Msg protocol
- Document model
- Tiled HTTP API



Duck Typing is a Good Idea

- "If it quacks like a duck..."
- Avoid `if type(x) == ...` unless you really, really mean it.
- When using `isinstance(x, ...)` use the broadest acceptable definition
 - `isinstance(x, list)`
 - `isinstance(x, collections, abc.Iterable)`



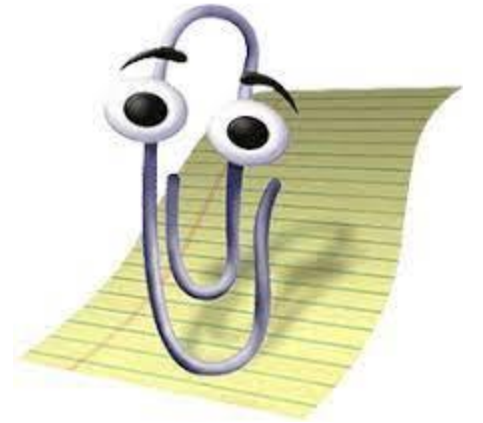
Don't be afraid to refactor or rewrite

- No code is ever right the first (or second) time.
- Refactoring the code once you understand the problem and the design trade-offs more fully helps keep the code maintainable.
- Version control, tests, and linting are your safety net, empowering you to make changes with confidence.











Permissiveness isn't always convenient

- It can be tempting to guess what the user means.
- This can result in deeply confusing bugs.
- Compromise:
 - Write the actual logic in strict code.
 - Write a thin friendly wrapper around it.
 - This enables writing *multiple* friendly wrappers with different opinions or defaults, targeting different types of users or use cases.



Write helpful error messages

- Be specific.
- Include what the wrong value was, what was wrong with it, and perhaps how it might be fixed.
- For example, if the code fails to locate a file it needs, it should say what it was looking for and where it looked.

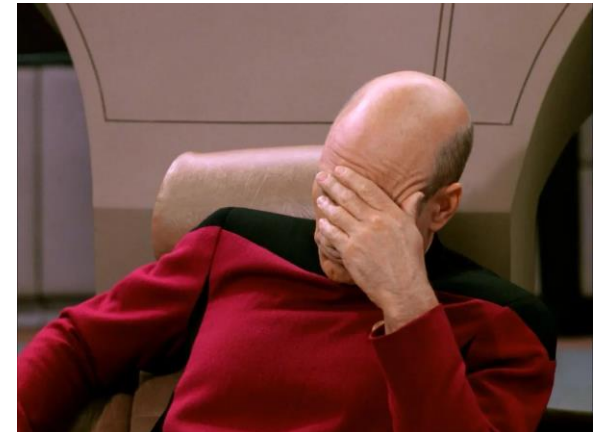
	Fill the water tank with fresh water to the MAX level indication.		The brew group is blocked by coffee powder. Clean the brew group.
	The bean hopper is empty. Put new coffee beans in the bean hopper.		Insert the drip tray and close the service door.
	The brew group is not in the machine or it is not inserted correctly. Insert the brew group.		The CoffeeSwitch is in the wrong position for the selected beverage. Push the lever down to the ESPRESSO position.
	The coffee grounds container is full. Make sure the machine is switched on. Then remove and empty the coffee grounds container.		The CoffeeSwitch is in the wrong position for the selected beverage. Push the lever up to the COFFEE position.

Keep I/O separate

- I/O functions should *only* do I/O
- Example: consume file path, return array and dictionary of metadata
- To support a new file format, it should not be necessary to touch the scientific logic
- The science code should operate on arrays and other data structures, never touch files directly

Avoid Changing State

```
# Bad!  
  
data = Data()  
  
data.load_data()  
  
data.prepare()  
  
data.do_calculations()  
  
data.plot()
```



Avoid Changing State

```
# Good!
```

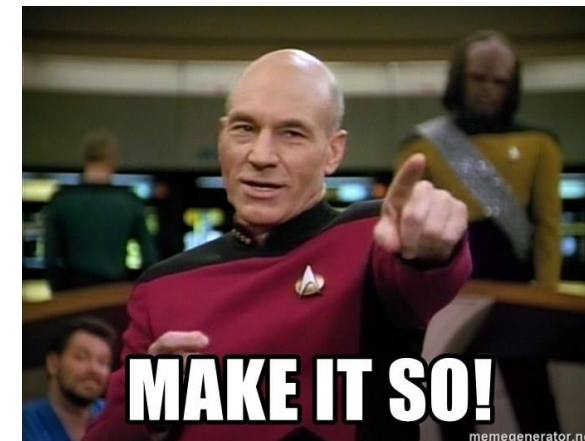
```
empty_data = EmptyData()
```

```
loaded_data = empty_data.load_data()
```

```
prepared_data = loaded_data.prepare()
```

```
computed_data = prepared_data.do_calculations()
```

```
computed_data.plot()
```



Avoid Changing State

```
# You can chain these for more succinct usage.  
computed_data =  
    EmptyData().load_data().prepare().do_calculations()  
computed_data.plot()
```

Consider: Can this just be a function?

- Object Oriented design has its place, but frequently does not add value in scientific code.
- Usually built-in or common types like list, dict, dataclass, or numpy array do the job and are easier to quickly understand.
- "It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures." – From ACM's SIGPLAN publication, (September, 1982), Article "Epigrams in Programming", by Alan J. Perlis of Yale University.

Optimize for readability

- Writing new code is easy. Maintaining code is hard.
- Code is read more times than it is written.
- Do not be too clever, or you may not be able to untie your own knots.
- Optimize for clarity over brevity.
- Use consistent and descriptive variable names.





Scientific Python Library Development Guide

This guide is maintained by the scientific Python community for the benefit of fellow scientists and research software engineers.

Start at the basics. Do you have a pile of scientific Python scripts or Jupyter notebooks that are becoming unwieldy? Are changes to some parts of your code accidentally breaking other parts of your code? Do you want to more maintainable, reusable, and shareable form? Start at the [tutorial](#).

Learn recommended tools and best practices. [Topical guides](#) provide task-based instruction on topics that scientists and research software engineers may encounter as their projects evolve and grow. This covers modern packaging([simple](#) or [compiled](#)), [style checking](#), [testing](#), [documentation](#), [static typing](#), [CI](#), and much more!

NEW PROJECT TEMPLATE

This guide comes with a [copier/cookiecutter/cruft](#) template for making new repos, [scientific-python/cookie](#). Eleven build backends including compiled backends, generation tested in Nox, and kept in-sync with the guide.

CHECKING AN EXISTING PROJECT

We provide [sp-repo-review](#), a set of [repo-review](#) checks for comparing your repository with the guidelines, runnable right in the guide via WebAssembly! All checks point to a linked badge in the guide.

Learn to write better research code. A high-level document on [principles](#) provides advice based on the community's collective experience building code that is easier for researchers to use successfully and easier to maintain over time.

Use our solutions for common tasks. A growing collection of [patterns](#) provides tested approaches for tasks and can be tricky to get exactly right, such as including data files with Python packages.

Related Resources

This guide does *not* cover the basics of Python itself or the scientific Python libraries; it focuses on making or maintaining a package. We recommend the [Scientific Python Lectures](#) if you want info.

This guide also does not cover version control, but it is essential to have a basic facility with git to use these tools successfully. We recommend the [Software Carpentry lesson on git](#).

<https://learn.scientific-python.org/development/>