Adoption of Python and modern software practices in high energy physics

Jim Pivarski

Princeton University – IRIS-HEP

September 8, 2022

Subject of this talk: a rough assortment of things

- Adoption of Python in high energy physics (HEP)
  - How Python is used in HEP, trend toward Python for data analysis
- Adoption of "modern software practices" (devops)
  - Source control: git/GitHub/GitLab
  - Automated tests: continuous integration (CI), continuous deployment (CD)
  - Modularity of packages: Lego-style analysis
  - Package management: pip, conda
  - Environment encapsulation: venv, conda, Docker
  - Social structures: silos versus mixing

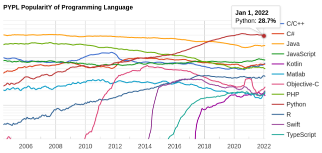
Is Python part of what I'm calling "modern"?

Python is leading every programming language popularity index.

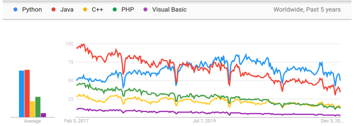
Tiobe

Jan 2022	Jan 2021	Change	Programming Language	Ratings	Change
1	3	^	Python	13.58%	+1.86%
2	1	v	C	12.44%	-4.94%
3	2	v	Java	10.66%	-1.30%
4	4		C++	8.29%	+0.73%
5	5		C#	5.68%	+1.73%
6	6		Visual Basic	4.74%	+0.90%
7	7		JavaScript	2.09%	-0.11%
8	11	^	Assembly language	1.85%	+0.21%

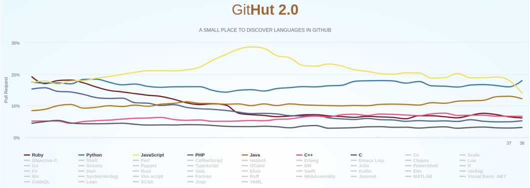
PYPL



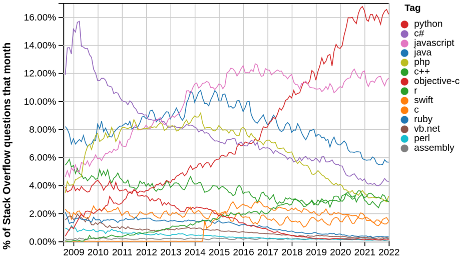
Google Trends



GitHut



StackOverflow



Is Python part of what I'm calling "modern"?

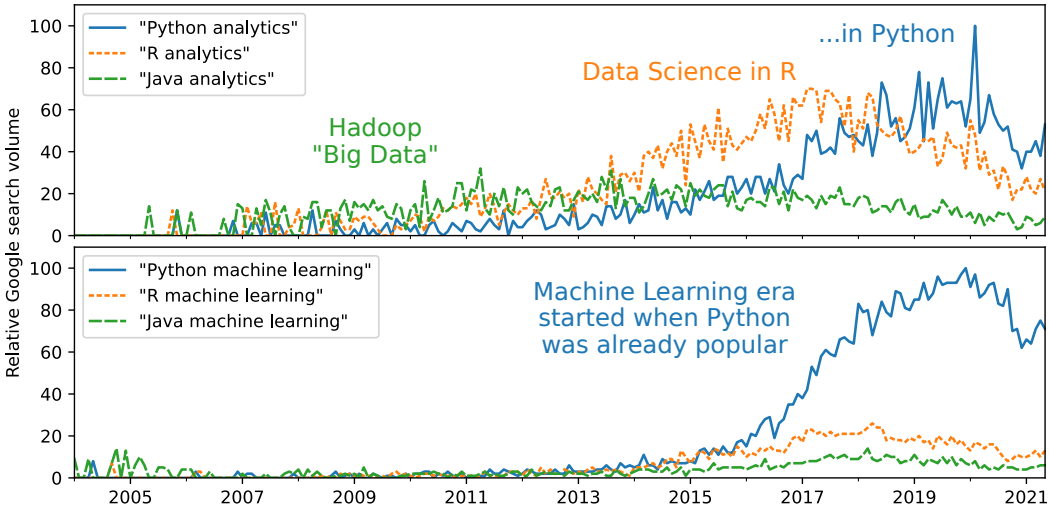
It's a recent thing, too.

Tiobe index's top ∼3 languages for the past 35 years:

1987	C, Lisp, Prolog	
1992	C, C++, Pascal	Python exists
1997	C, C++, (Visual) Basic	Python is #28
2002	Java, C, C++, (Visual) Basic	Python is #12
2007	Java, C, C++, (Visual) Basic	Python is #7
2012	Java, C, C++, C#	Python is #8
2017	Java, C, C++, C#	Python is #5
2022	Python, C, Java, C++, C#	Python is #1

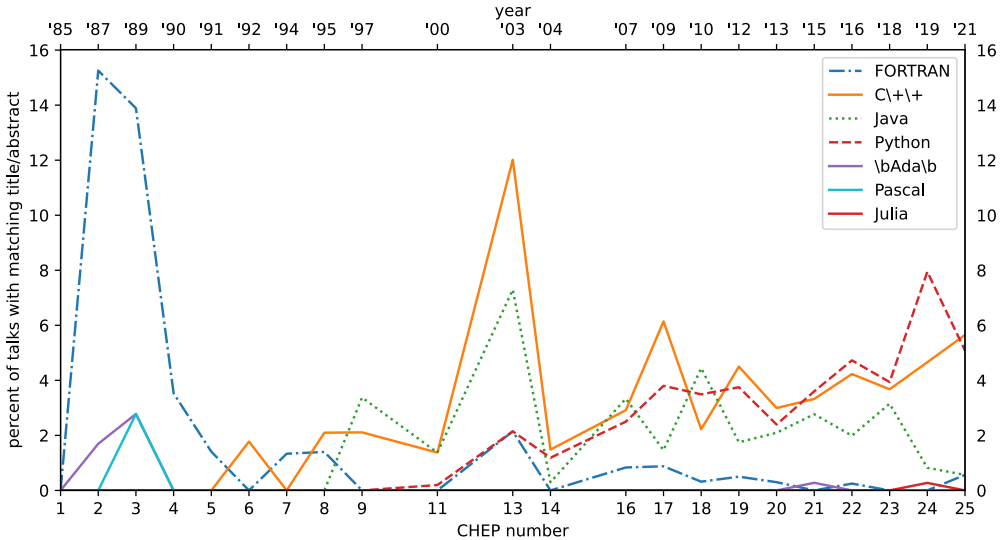
Not all programmers are data analysts: what about us?

Google search trends for language name + "analytics" or "machine learning."



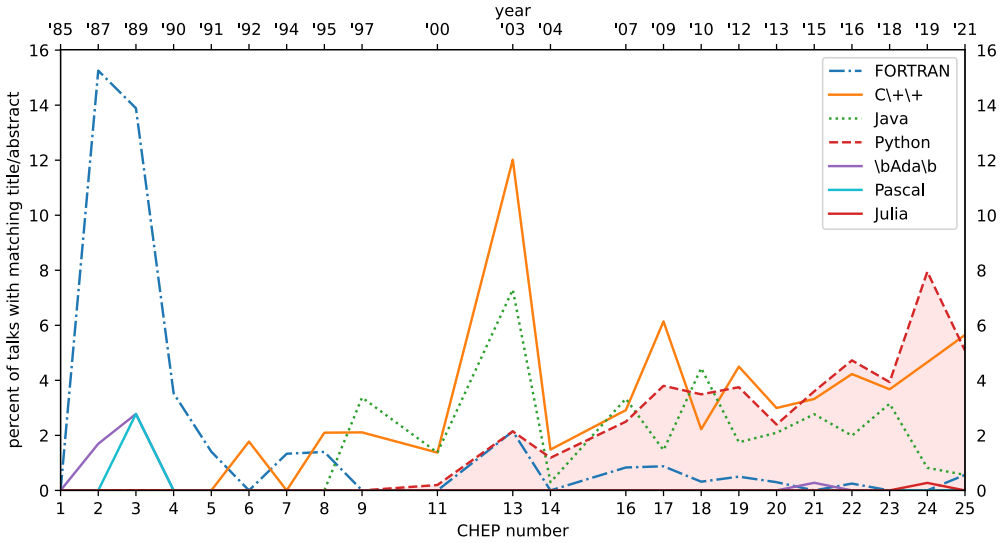
Not all data analysts are physicists: what about us?

Regex matches to all Computing in High Energy Physics (CHEP) titles and abstracts.



Not all data analysts are physicists: what about us?

Regex matches to all Computing in High Energy Physics (CHEP) titles and abstracts.



Emerging Standard ? Python as "Software Glue"

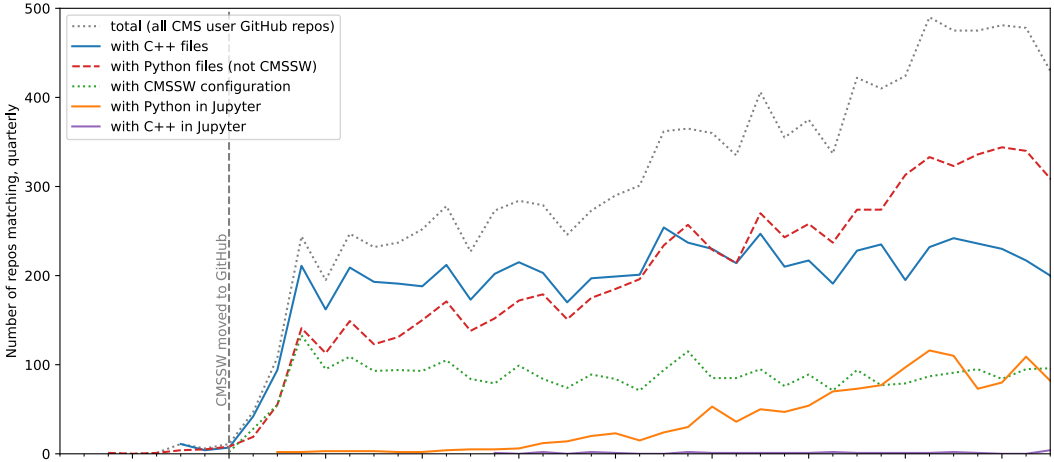
- Clear trend towards Python
  - Used by: ATLAS (Athena),CMS, D0, LHCb (Gaudi), SND,...
  - Used by: Lizard/Anaphe, HippoDraw, JAS (Jython)...
  - Architecturally, scripting is "just another service"
  - ROOT is the exception to the "Python rule"
    - CINT interpreter plays a central role
    - Developers and users seem happy
- Python is popular with developers...
  - Rapid prototyping; gluing together code
  - (Almost) auto-generation of wrappers (SWIG)
- ...but acceptance by users not yet proven
  - Another language to learn, syntax,...

But note the emphasis on “glue” and “configuration,”
not the process of analyzing data.

How do physicists *use* Python?

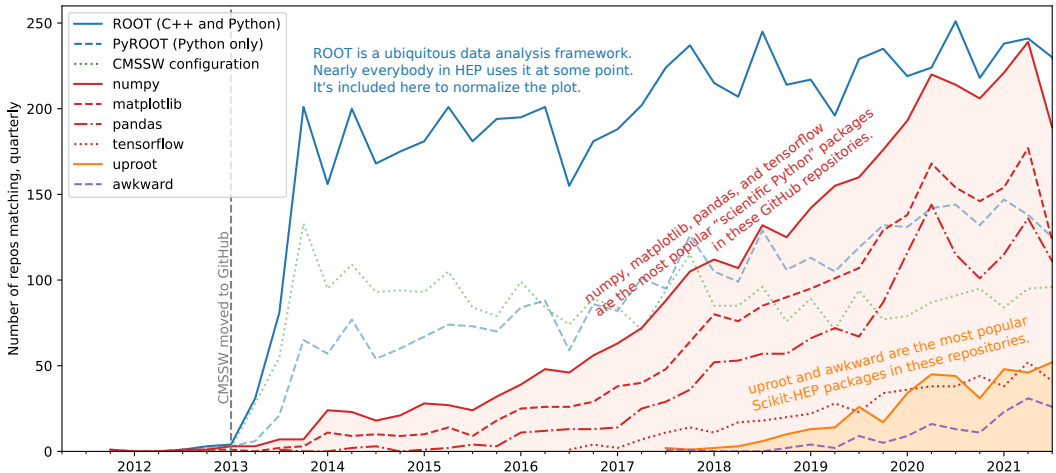
Analysis of 11 635 GitHub repos created by 2 172 physicists

Software for the CMS experiment is on GitHub and all CMS physicists need to fork it. Using that, we can identify all of those physicists' non-fork, public repositories.

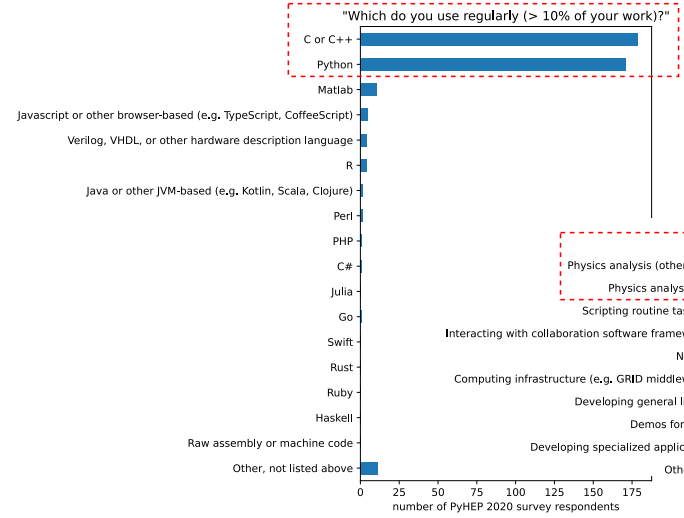


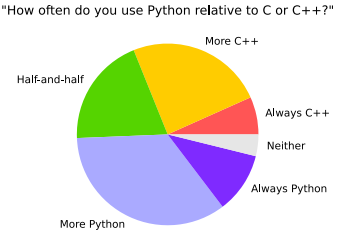
Analysis of 11 635 GitHub repos created by 2 172 physicists

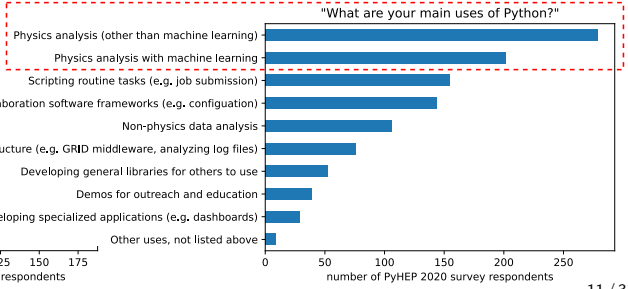
By regex searching for "`import [A-Za-z_][A-Za-z_0-9]*`" etc., we can count the number of physicist repositories that use different packages over time.



You can also just ask people: PyHEP 2020 survey ($N = 406$)







There has been a massive transition to Python for HEP analysis.

Why does it matter?

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.
- C++: performance and software architecture are up-front concerns; favored among large-scale HEP processing, high-speed trading, gaming engines.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.
- C++: performance and software architecture are up-front concerns; favored among large-scale HEP processing, high-speed trading, gaming engines.
- Java and Go: software architecture is an overriding concern, especially in networked services; backend web engineers and internal corporate networks.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.
- C++: performance and software architecture are up-front concerns; favored among large-scale HEP processing, high-speed trading, gaming engines.
- Java and Go: software architecture is an overriding concern, especially in networked services; backend web engineers and internal corporate networks.
- Rust: performance with memory safety overrides all.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.
- C++: performance and software architecture are up-front concerns; favored among large-scale HEP processing, high-speed trading, gaming engines.
- Java and Go: software architecture is an overriding concern, especially in networked services; backend web engineers and internal corporate networks.
- Rust: performance with memory safety overrides all.
- Julia: academic/scientist community, but very focused on performance.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.
- C++: performance and software architecture are up-front concerns; favored among large-scale HEP processing, high-speed trading, gaming engines.
- Java and Go: software architecture is an overriding concern, especially in networked services; backend web engineers and internal corporate networks.
- Rust: performance with memory safety overrides all.
- Julia: academic/scientist community, but very focused on performance.
- Haskell: theoretical aspects of computing; computer scientists.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

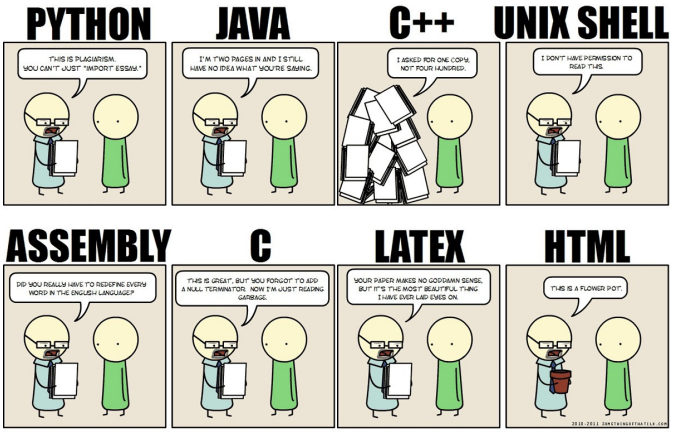
Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.

- Python: culture of readability first, performance and software architecture later, if at all. More focus on the domain (e.g. science) than programming.
- C++: performance and software architecture are up-front concerns; favored among large-scale HEP processing, high-speed trading, gaming engines.
- Java and Go: software architecture is an overriding concern, especially in networked services; backend web engineers and internal corporate networks.
- Rust: performance with memory safety overrides all.
- Julia: academic/scientist community, but very focused on performance.
- Haskell: theoretical aspects of computing; computer scientists.
- C#: Windows. Swift: Mac.

There has been a massive transition to Python for HEP analysis.

Why does it matter?

Humans use programming languages to communicate technical issues with each other as much as with computers. Cultures develop around them.



There has been a massive transition to Python for HEP analysis.

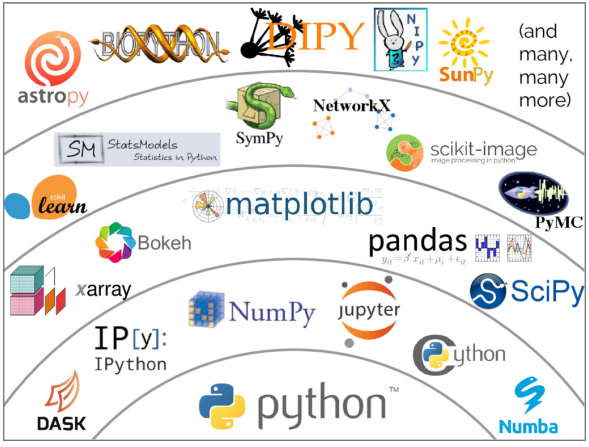
Why does it matter?

We are in closer communication with developers of Python open source projects like NumPy, Scikit-Learn, and AstroPy.

Large-scale HEP processing frameworks are no longer the default model for how to develop a data analysis package.

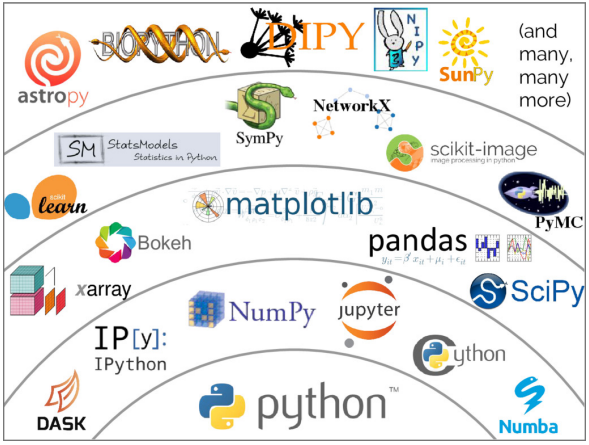
Model: small, specialized packages for Lego-style data analysis

Scientific Python ecosystem

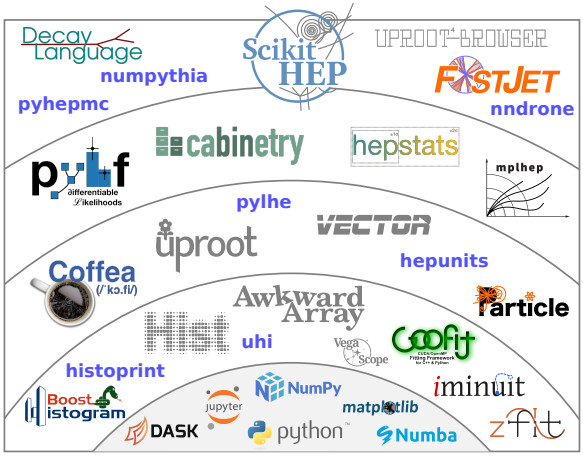


Model: small, specialized packages for Lego-style data analysis

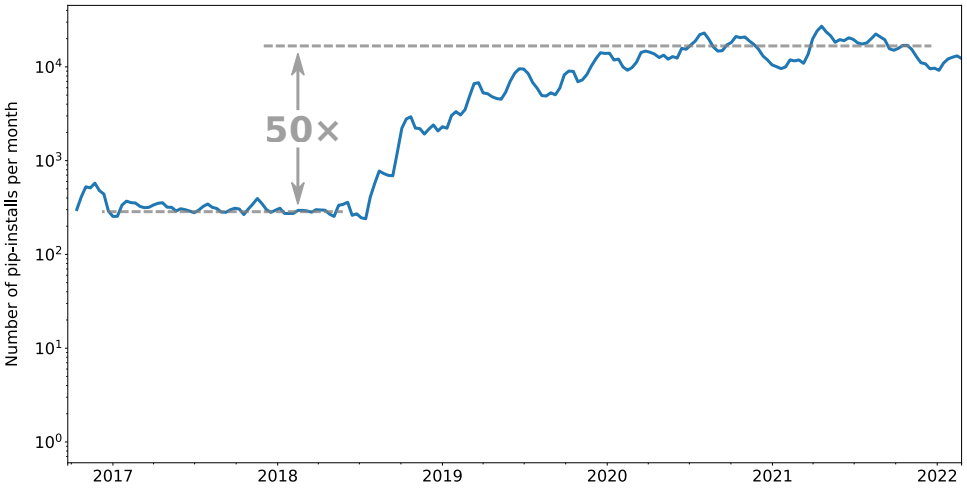
Scientific Python ecosystem



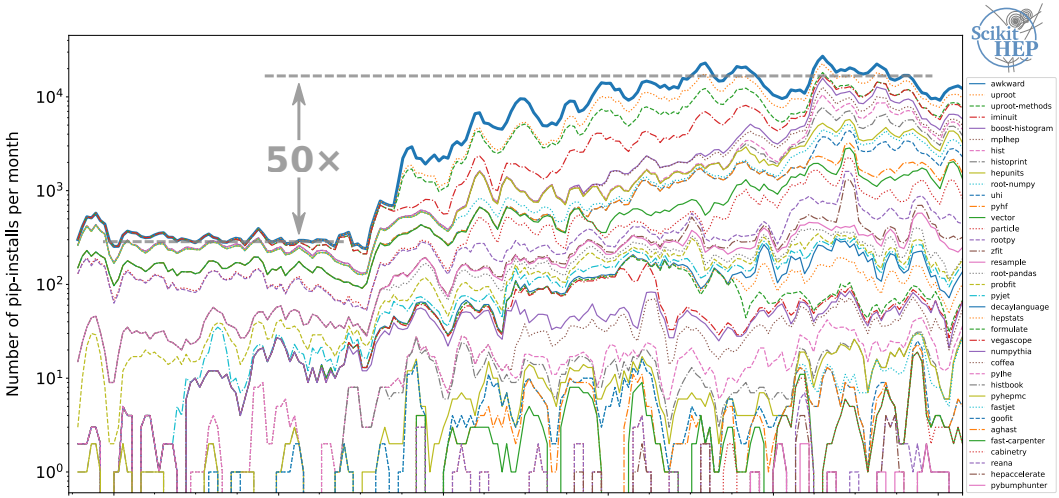
Scikit-HEP ecosystem



Download rates of Scikit-HEP packages (on Mac & Windows)



Download rates of Scikit-HEP packages (on Mac & Windows)



There isn't a central authority to divide up the space of analysis problems and assign people to develop packages.

But interested developers find underserved areas and build them up. Borders between them start vague and eventually get defined.

Case study: development of histogram packages

The scientific Python world lacked HEP-style histograms; it's one of the things we had to make ourselves.

Case study: development of histogram packages

The scientific Python world lacked HEP-style histograms; it's one of the things we had to make ourselves.

Also, it seems easy: just bin and count, right?

Case study: development of histogram packages

The scientific Python world lacked HEP-style histograms; it's one of the things we had to make ourselves.

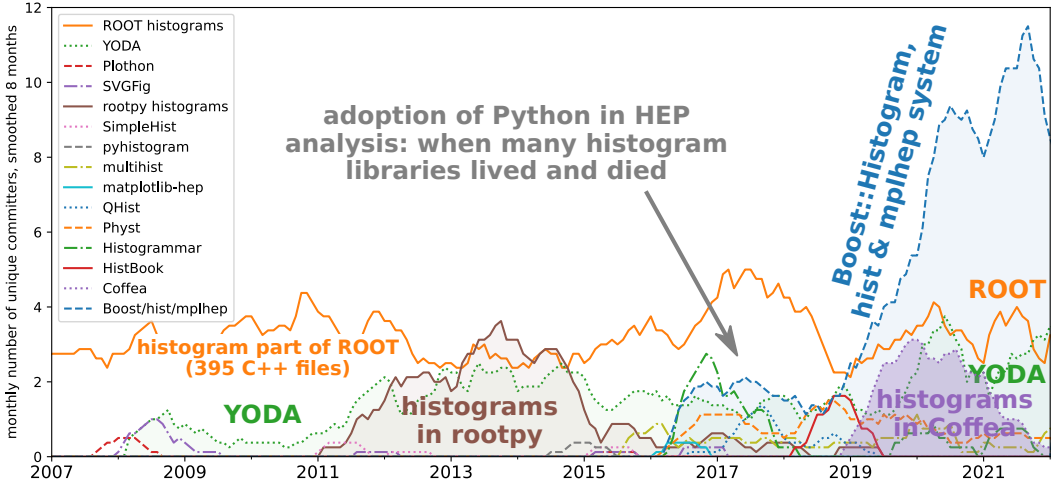
Also, it seems easy: just bin and count, right?

Physicists have created at least 20 histogram packages in Python, mostly by single authors.

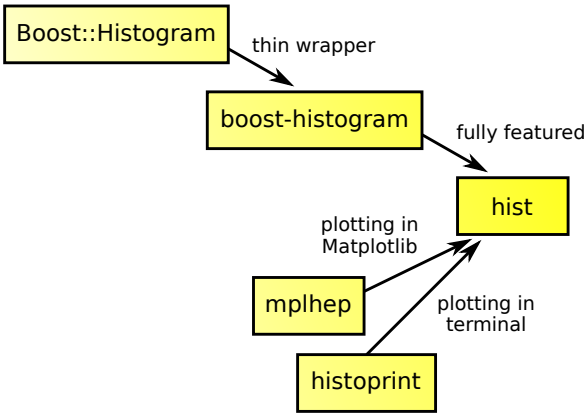
- PyROOT (2004–now)
- PAIDA (2004–2007)
- Plothon (2007–2008)
- SVGFig (2008–2009)
- YODA (2008–now)
- DANSE (2009–2011)
- rootpy (2011–2019)
- SimpleHist (2011–2015)
- pyhistogram (2015)
- multihist (2015–now)
- matplotlib-hep (2016)
- QHist (2017–2019)
- Physt (2016–now)
- Histogrammar (2016–now)
- HistBook (2018–2019)
- Coffea.hist (2019–2022)
- boost-histogram (2019–now)
- mplhep (2019–now)
- histoprint (2020–now)
- hist (2020–now)

Histogram proliferation and convergence

Number of unique developers contributing to each package per month (in git).
Measures concentration of *developer activity*, not users.

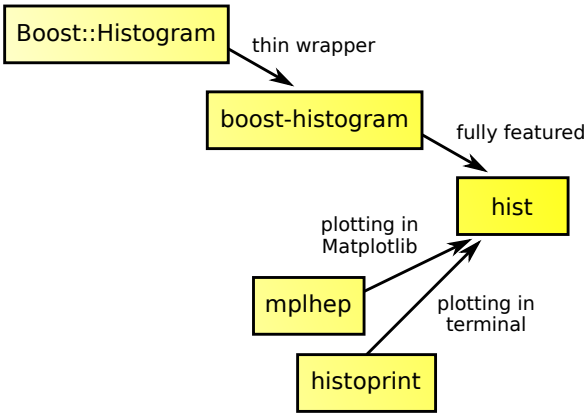


Boost::Histogram, hist & mplhep “system”



Originally, each of these was developed independently by a single author.

Boost::Histogram, hist & mplhep “system”

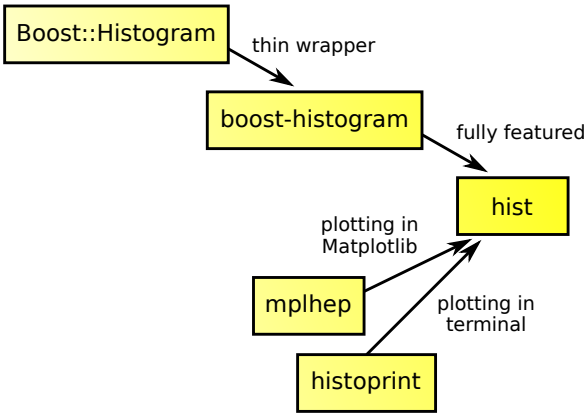


Originally, each of these was developed independently by a single author.

They each provide a piece of functionality users can get through

```
import hist
```

Boost::Histogram, hist & mplhep “system”



Originally, each of these was developed independently by a single author.

They each provide a piece of functionality users can get through

```
import hist
```

Now, 47 developers have contributed to these packages, and 20 contributed to more than one.

Now we define the borders: agreed-upon protocols

uhi 0.3.1 documentation

Search the docs ...

UHI: Unified Histogram Interface

CONTENTS:

Indexing

Indexing+

Plotting

Help for plotters

The module `uhi.numpy_plottable` has a utility to simplify the common use case of accepting a PlottableProtocol or other common formats, primarily a NumPy `histogram`/`histogram2d`/`histogramdd` tuple. The `ensure_plottable_histogram` function will take a histogram or NumPy tuple, or an object that implements `.to_numpy()` or `.numpy()` and convert it to a `NumPyPlottableHistogram`, which is a minimal implementation of the Protocol. By calling this function on your input, you can then write your plotting function knowing that you always have a `PlottableProtocol` object, greatly simplifying your code.

The full protocol version 1.2 follows:

(Also available as `uhi.typing.plottable.PlottableProtocol`, for use in tests, etc.

```
"""
Using the protocol:

Producers: use isinstance(myhist, PlottableHistogram) in your tests; part of
the protocol is checkable at runtime, though ideally you should use MyPy; if
your histogram class supports PlottableHistogram, this will pass.

Consumers: Make your functions accept the PlottableHistogram static type, and
MyPy will force you to only use items in the Protocol.
"""
```

Contents

For users, the multiplicity of packages isn't a problem because package managers resolve dependencies.

```
pip install hist
```

For developers, it provides more visibility,
ease of maintenance, and protection against scope-creep.

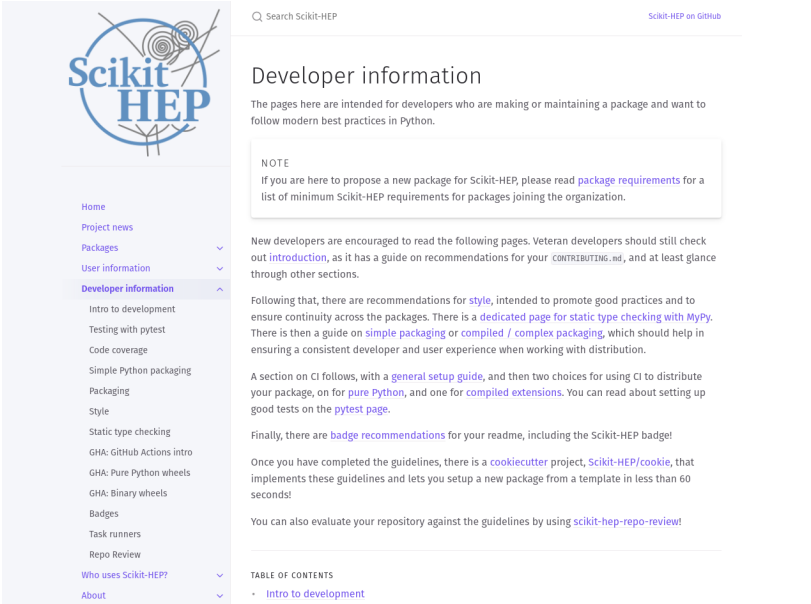
Small-scoped packages are not the only cultural shift

Various "devops" practices are common in scientific Python tool development, and promoted among Scikit-HEP members.

- Automated test suites, checked on every git commit
- Encapsulation of test environment and matrix of platforms to test
- Automated (push button) deployment to PyPI and conda-forge
- Issue tracking, pull request workflow
- Pull request reviews, extensive discussions on GitHub
- Contributing to more than one's "own" project

These are now cultural norms. Note that some graduate student developers were born after the *Manifesto for Agile Software Development* (2001). :)

Scikit-HEP developer pages: recommendations and cookiecutter



Hopefully, I didn't paint too rosy of a picture: there are issues.

Hopefully, I didn't paint too rosy of a picture: there are issues.

Many of our developers are self-selected from advanced data analysts, and the tools they write tend to be expert-oriented.

Hopefully, I didn't paint too rosy of a picture: there are issues.

Many of our developers are self-selected from advanced data analysts, and the tools they write tend to be expert-oriented.

Python itself is beginner-friendly, but analysis tools/docs aren't always.

Hopefully, I didn't paint too rosy of a picture: there are issues.

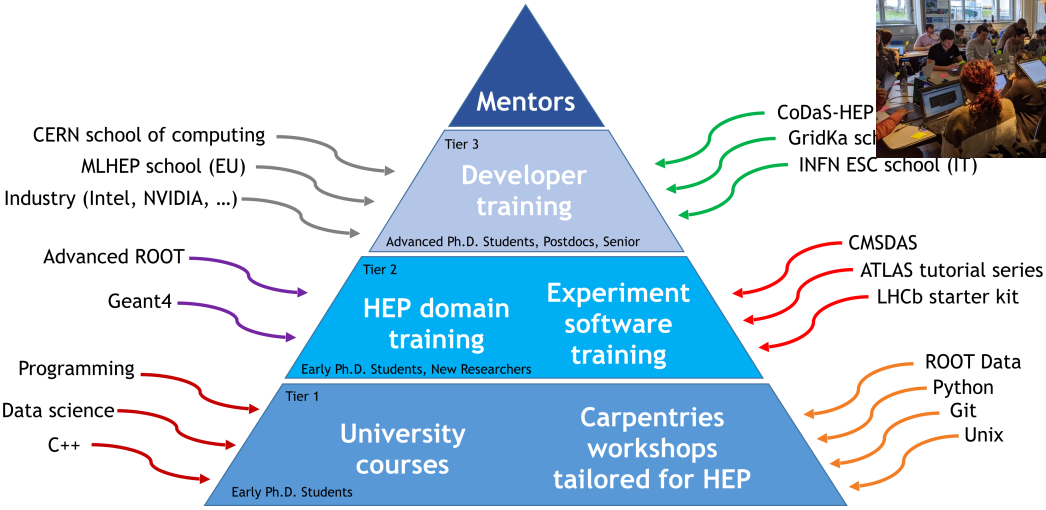
Many of our developers are self-selected from advanced data analysts, and the tools they write tend to be expert-oriented.

Python itself is beginner-friendly, but analysis tools/docs aren't always.

Also, there isn't a clearly centralized place to ask questions: there are too many places to ask questions.

There are, however, extensive projects to improve training

One of IRIS-HEP's projects is to develop HEP software training, modeled on and in collaboration with Software Carpentries.

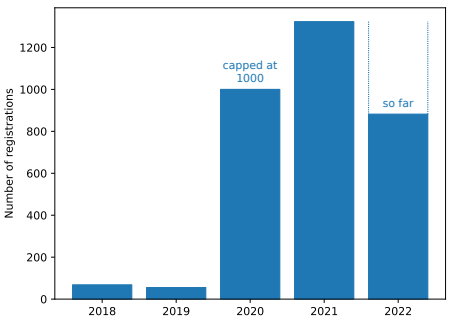


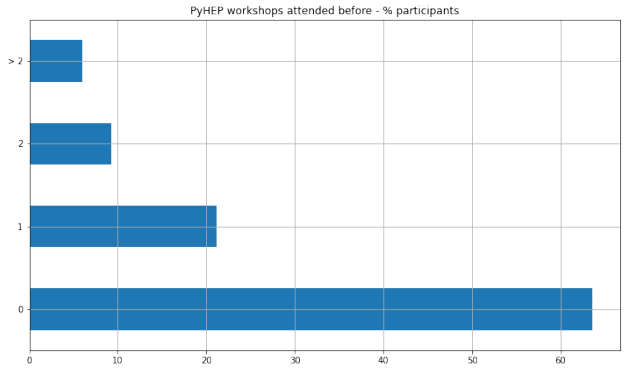
PyHEP workshop evolved from developer-focus to a training event

2018

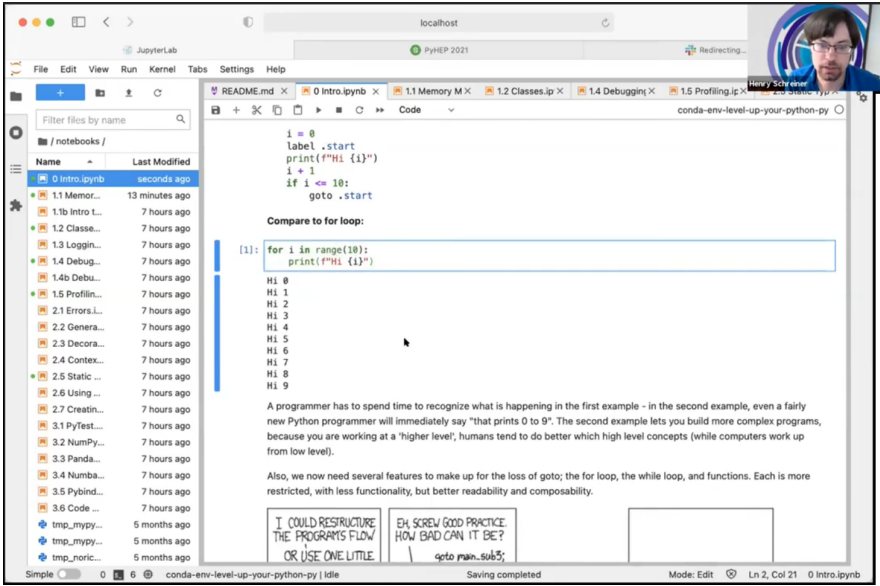
2021

Can you guess what happened?

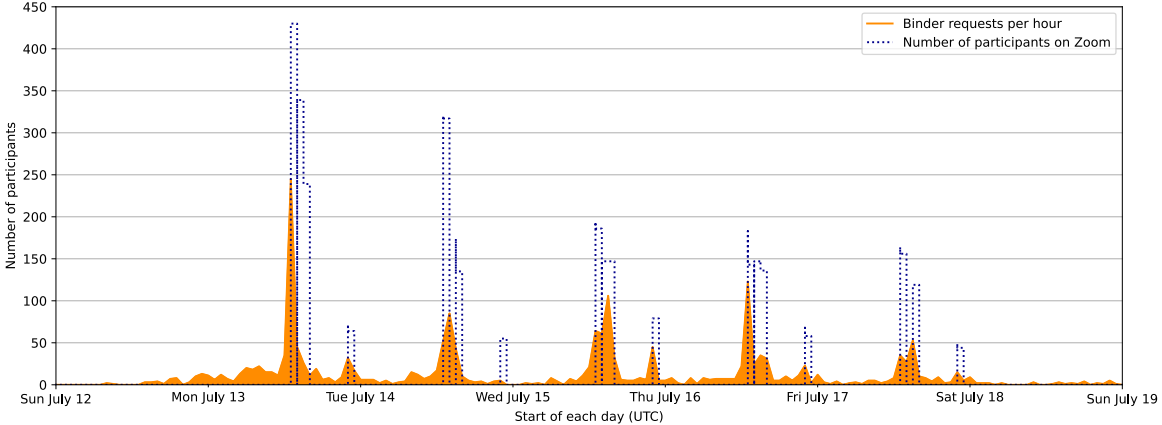




Most presentations/tutorials are in computable notebooks



And 20–40% of the 1000 registrants attend/run the notebooks



PyHEP 2022 is next week, September 12–16

PyHEP 2022 (virtual) Workshop

Sep 12 – 16, 2022
Europe/Zurich timezone

Enter your search term

- Overview
- Call for Abstracts
- Timetable
- Registration
- Participant List
- Videoconference
- Proceedings
- Code of conduct
- EDI statement

Contact us

pyhep2022-organisation...

The **PyHEP workshops** are a series of workshops initiated and supported by the HEP Software Foundation (HSF) with the aim to provide an environment to discuss and promote the usage of Python in the HEP community at large. Further information is given on the PyHEP Working Group website.

PyHEP 2022 will be a *virtual* workshop. It will be a forum for the participants and the community at large to discuss developments of Python packages and tools, exchange experiences, and inform the future evolution of community activities. There will be ample time for discussion.

The agenda is composed of plenary sessions:

1) Hands-on tutorials.
2) Topical sessions.
3) Presentations following up from topics discussed at PyHEP 2021.

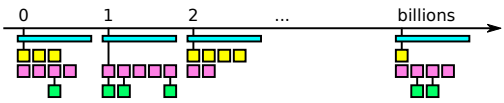
Registration is open until September 16th. There are *no workshop fees*.

You are encouraged to register to the PyHEP WG Gitter channel and/or to the HSF forum to receive further information concerning the organisation of the workshop. Workshop updates and information will also be shared on the workshop Twitter in addition to email. Follow the workshop @PyHEPConf.

Last slide: my own project

Awkward Array

An array library for **nested, variable-sized data**, including arbitrary-length lists, records, mixed types, and missing data, using **NumPy-like idioms**.



Arrays are **dynamically typed**, but operations on them are **compiled and fast**.

Example of an “awkward” array

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

Record structures with differently typed fields

Array of variable-length lists (“ragged” or “jagged” arrays)

Nested variable-length lists

Missing data

NumPy-like expression

```
output = np.square(array["y", ..., 1:])
```

Result

```
output.to_list()
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

1.5 seconds
2.1 GB of memory

Equivalent Python

```
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(np.square(number))
        tmp1.append(tmp2)
    output.append(tmp1)
```

Heterogenous data (union/variant types)

140 seconds
22 GB of memory

How it works

An entire array consists of **one small tree** with large, contiguous data buffers attached to each node (color coded with the above).

Compiled operations are performed on these data buffers, not the objects they represent.

(research scientists)

Jim Pivarski
Princeton University

Henry Schreiner
Princeton University

(research software engineers)

Ianna Osborne
Princeton University

Ioana Ifrim
Princeton University

(postdoc)

Angus Hollands
University of Birmingham → Princeton University

(undergraduates)

Anish Biswas
Manipal Institute of Technology

Manasvi Goyal
Delhi Technological University

Aryan Roy
Manipal Institute of Technology

(contractors)

Douglas Davis
Anaconda, Inc.

Martin Durant
Anaconda, Inc.

24 more contributors on GitHub