A brief introduction to scientific Python

and why it has proliferated to so many fields

Chase Coleman and John Stachurski

March 2024

Goal of these slides

My goal in these slides is convince you that Python is (one of) the right tools to be using for economics (and even for science more generally).

I pulled (sometimes heavily) from a great talk, titled "The Unexpected Effectiveness of Python in Science", given by Jake VanderPlas at PyCon in May 2017. You can find a video of his talk on YouTube

Where did Python start

Originally developed by Guido van Rossum in the late 1980s/early 1990s. It was originally developed as a hobby project with the intention of being focused on being a "teaching language".

Core Python philosophy:

Beautiful is better than ugly
Explicit is better than implicit
Simple is better than complex
Complex is better than complicated
Readability counts

Which fields use Python?

- Astronomy
- Biology
- Climatology
- Computer science
- Engineering
- Environmental science
- Physics
- Statistics

... and, increasingly, our own field of economics!

What makes Python popular amongst such a broad number of scientists?

- Easy to learn
- A focus on readability – Readability makes it to easier collaborate (and review!)
- Interoperability with other languages
- Large ecosystem of libraries with "investment" into this ecosystem from large companies
- Open ethos

Easy to learn / Readable

- Python was designed to be easy to learn from the start.
- Very high level – The language gets out of your way and effectively allows you to write (and run) pseudo-code
- Easy to read (forces you to space out your code) which makes reviewing code from others less of a chore

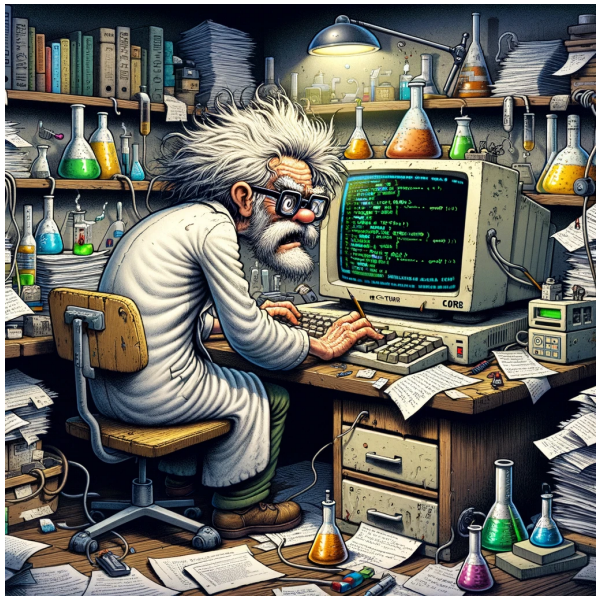
Interoperability / Ecosystem

- The fact that it is interpreted makes it easy to share with others – Check out the gravity waves notebook!
- Lots of great tools for running things like notebooks! Colab, JupyterLite, etc...
- Significant amounts of collaboration/investment between large companies/universities

Open ethos

- The Python community is very focused on "openness"
- The scientific community, including us within economics, should care about openness because it allows us to make more progress
- Some great initiatives for openness include QuantEcon, Policy Simulation Library, etc...

A short story: Part 1



Everyone, meet Bob. Bob was once a Fortran-only programmer at CERFACs.

He was responsible for working on a tool called Projector which he had originally developed in Fortran in 2010.

A short story: Part 2

As new members joined his team, none of them wanted to work with Projector because they didn't understand his code and, even when they did help, the quality of their Fortran code was a disappointment to him (and so he would often have to re-do what they had contributed).

Bob saw Python as a hassle, an indentation-greedy language supported by many impolite youngsters who were too happy to bully others by saying "this is not Pythonic" with the penetrating stare of those who saw the light. In short, Bob was annoyed that people used Python...

A short story: Part 3

Bob was eventually convinced that it would be worth seeing what would happen if he and the "youngsters" rewrote his Fortran code in Python so that more people on the team could be involved in the process (and, along the way, he hoped to convert them to a "real" programming language).

As Bob and his collaborators rebuilt Projector in Python, he discovered that he could experiment with different algorithms and data representations – Often with a cost as small as changing a few lines of code or the argument to a function call.

As he was working on the Python code, he read some Scipy documentation and realized that it might be better if he used a KD tree to represent the data he was operating on.

A short story: Part 4

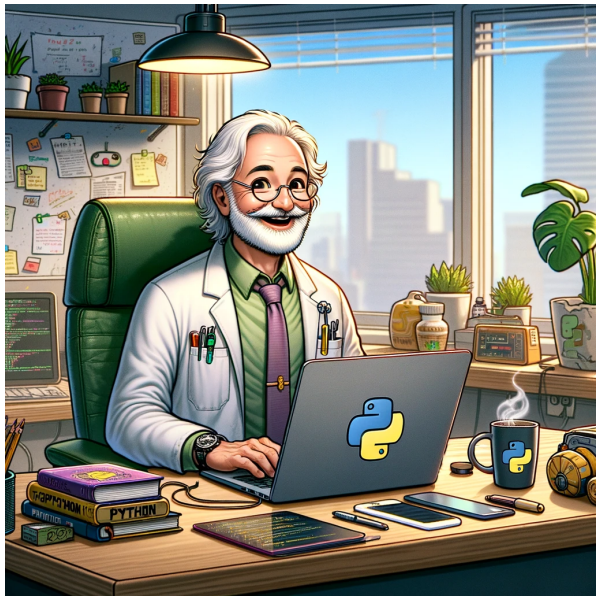
Finally, after a few weeks of work, the moment of truth came. Bob had experimented with relatively small examples on his local computer and the Fortran and Python versions of Projector seemed to line up.

Bob chose to perform a comparison on a relatively large sample (thinking that it would break the Python code and his Fortran code would reign supreme).

The Fortran code ran in 6 hours and 30 minutes – Pretty good for the size of problem he had designed.

Next up, the Python code. The Python code ran in 4 minutes – The Python code ended up faster because it allowed Bob and his team to select better ways to structure the data and a wider variety of algorithms (that they didn't have to code from scratch).

A short story: Finale



In this moment, Bob became enlightened and never wrote Fortran code again.

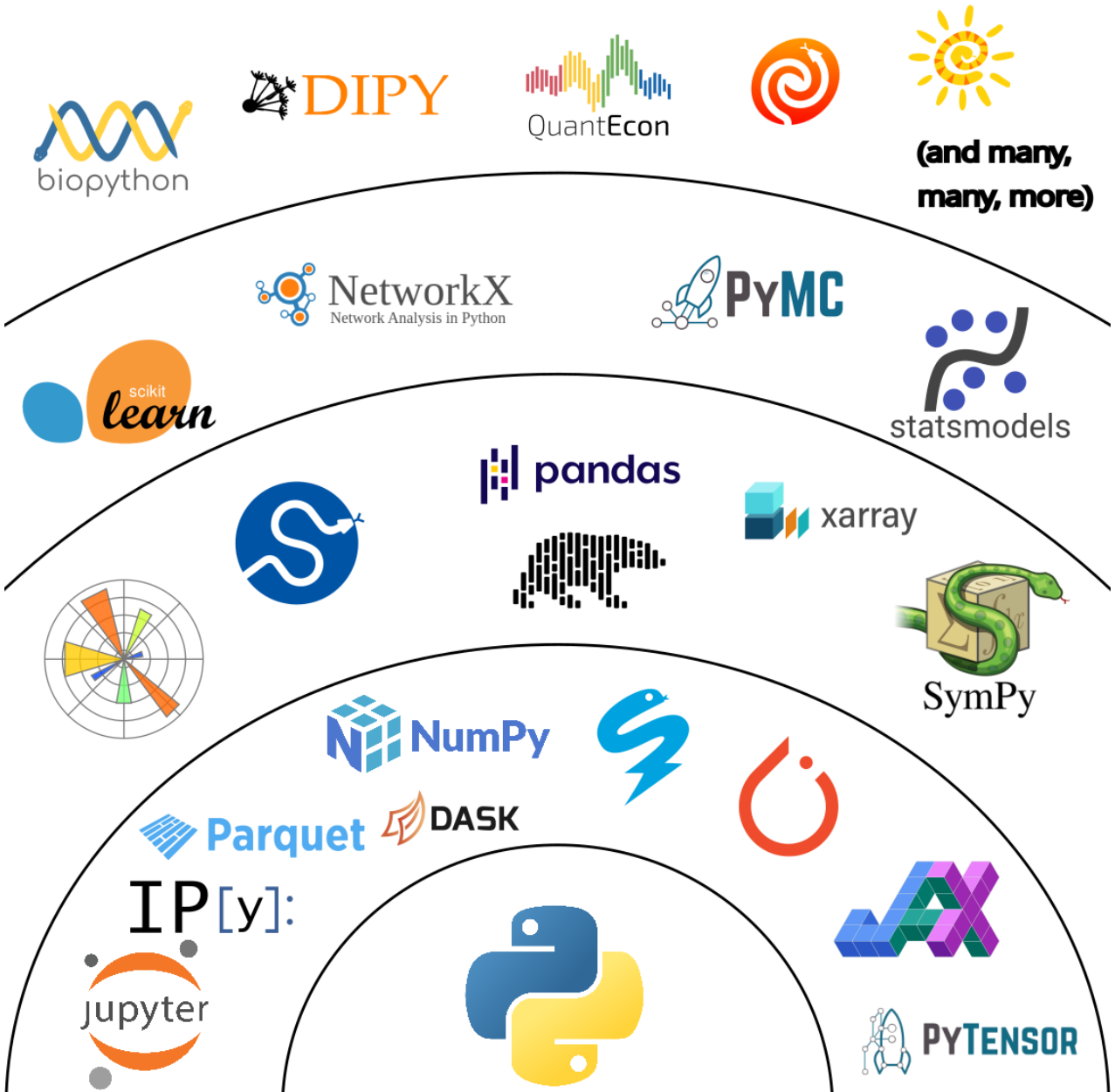
Borrowed from a blog post by Centre Europeen de Recherche et Formation Avancee en Calcul Scientifique

The layers of scientific Python

I view there being 5 main layers of scientific Python

1. Core Python
2. Numeric infrastructure
3. Scientific infrastructure
4. General scientific tools
5. Domain specific tools

Scientific Python landscape



Scientific Python: Core Python

The entire scientific stack in Python begins with "core Python" which includes the standard library of Python packages.

```
import math

x = 2 * math.pi
y = 1.0 / x * math.sin(x)
```

Scientific Python: Numeric infrastructure

The numeric infrastructure stack expands what is possible within Python by adding additional types and packages that you can directly build on. This will include things like NumPy's array type

```
import math

import numpy as np

x = np.linspace(0.01, 2 * math.pi, 50)
y = 1.0 / x * np.sin(x)

x@y
```

Scientific Python: Scientific infrastructure

The scientific infrastructure differs from numeric infrastructure because, rather than build general numeric tools, it focuses on tools to perform specific actions, such as interpolation or optimization.

This infrastructure often provides the type of "algorithm legos" that makes it easy to experiment with different algorithms as you solve your problem of interest.

Scientific Python: Scientific infrastructure

```
import math

import numpy as np
import scipy.interpolate as interp

x = np.linspace(0.01, 2 * math.pi, 50)
y = 1.0 / x * np.sin(x)

pwci = interp.CubicSpline(x, y)
pwci(math.pi)
```

Scientific Python: General scientific tools

General scientific tools are about simplifying the code to do a specific but widely used task. For example, a linear regression really should only take a single line of code.

- If you used numeric infrastructure, it might take a couple hundred lines of code
- If you used the scientific infrastructure, it might take 10-20 lines of code

Scientific Python: General scientific tools

```
import numpy as np
from sklearn.linear_model import LinearRegression

x = np.arange(0, 100, 50)
y = 2*x + 0.1*np.random.randn(50)

reg = LinearRegression().fit(x, y)
```

Scientific Python: Domain specific tools

The final layer of the scientific Python stack is the tools used for domain expecific analysis.

For example, the code inside of the QuantEcon Python package is all domain specific code and is meant to facilitate the type of analysis/modeling that is done by economists.