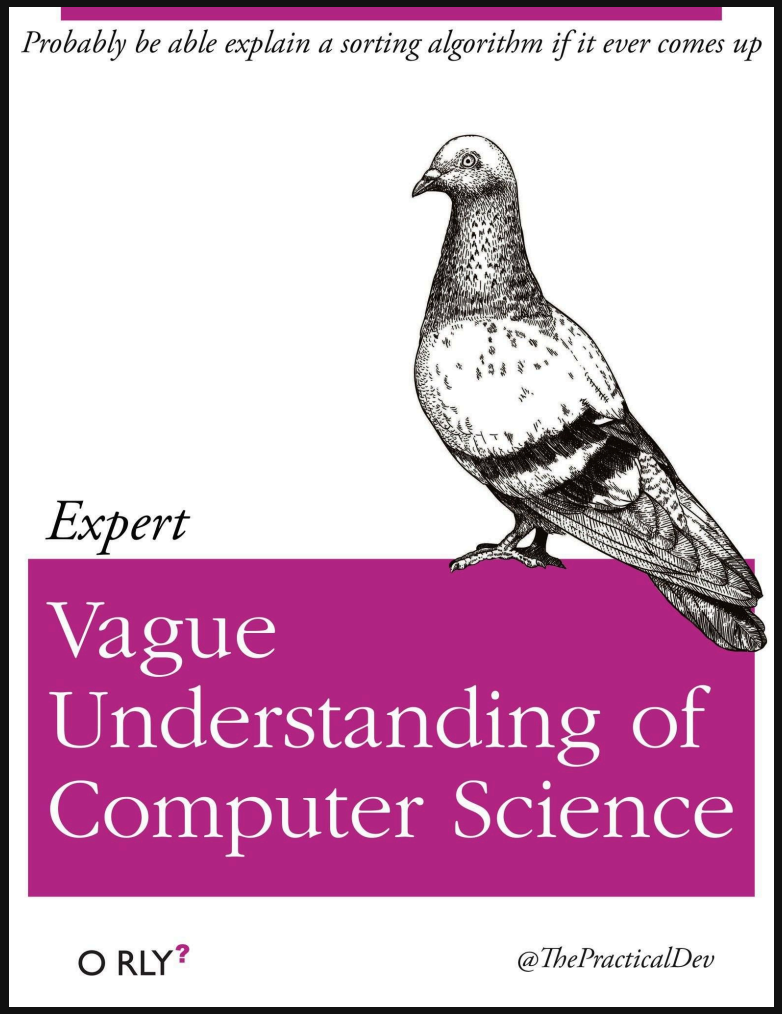
Programming with Python

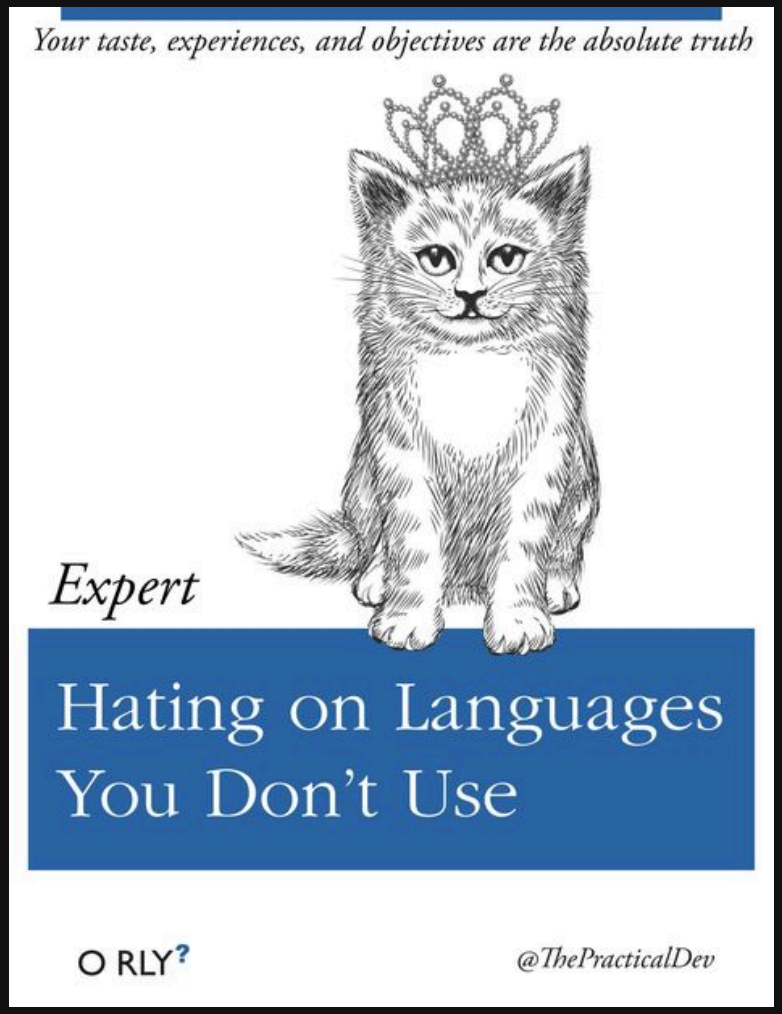
Ecosystem

PROBLEMS

Simplicity


Probably be able explain a sorting algorithm if it ever comes up
Expert
Vague
Understanding of
Computer Science
O RLY?
@ThePracticalDev

Other Language



NAMESPACE

dir

```
print(s:=set(dir()))
print(set(dir()) - s)
a = 5
print(set(dir()) - s)
```

```
{'__builtins__', '__spec__', '__doc__', '__cached__', '__name__', '__packa
{'s'}
{'a', 's'}
```

scope

```
price = 100  # 🌍

def calculate():
    price = 50  # 📦 Creates new local var
    print(price)

calculate()
print(price)
```

```
50
100
```

scope

```
price = 100  # 🌍

def calculate():
    global price
    price = 50  # 📦 Creates new local var
    print(price)

calculate()
print(price)
```

```
50
50
```

scope resolution

LEGB : Local → Enclosing → Global → Built-in

```
scope = 'global 🌍'

def func():
    # scope are restricted to function encloser
    scope = 'enclosing 📦'
    def nested_func():
        scope = 'local 🎁'
        print(scope)

    nested_func()
    print(scope)

func()
print(scope)
```

```
local 🎁
enclosing 📦
global 🌍
```

builtins

```
print('max is ', max)

max = 10
print('max is ', max)
print('max is ', __builtins__.max)

del max
print('max is ', max)
```

```
max is  <built-in function max>
max is  10
max is  <built-in function max>
max is  <built-in function max>
```

MODULES

batteries included 🔋

```
help('modules')  # list all the modules
```

import

```python
import os
print(os)

import sys
print(sys)

import math
print(math)
```

```
<module 'os' (frozen)>
<module 'sys' (built-in)>
<module 'math' from '/usr/lib/python3.13/lib-dynload/math.cpython-313-x86_
```

inside module

```python
import os

print(dir(os))
# help(os)
```

```
['CLD_CONTINUED', 'CLD_DUMPED', 'CLD_EXITED', 'CLD_KILLED', 'CLD_STOPPED',
```

docs

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is an active collection of hundreds of thousands of components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

module

```python
import os
print(os.cpu_count())

from os import cpu_count
print(cpu_count())

from math import pi, tau, e
print(pi, tau, e)

from sys import *
print(platform)
```

```
12
12
3.141592653589793 6.283185307179586 2.718281828459045
linux
```

CREATE-MODULE

why

- Code reuse ♻️
- System namespace partitioning
- Implementing shared services or data

naming

file type	plain text
file name	`^[_a-zA-Z][_a-zA-Z0-9]*.py$`
extensions	.py

import

```python
print('i am from a.py')
var = 5
```

```python
import a
print(a.var)
print(dir(a))
```

```
i am from a.py
5
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__nam
```

variables

```
if __name__ == '__main__':
    print('something')
```

```
something
```

PYTHONPATH

```
python -c "import sys; print(*sys.path, sep='\n')"
```

```
/usr/lib/python313.zip
/usr/lib/python3.13
/usr/lib/python3.13/lib-dynload
/usr/lib/python3.13/site-packages
```

```
mkdir --parent /tmp/path/to/module
export PYTHONPATH+=':/tmp/path/to/module'
python -c "import sys; print(*sys.path, sep='\n')"
```

```
/tmp
/tmp/path/to/module
/usr/lib/python313.zip
/usr/lib/python3.13
/usr/lib/python3.13/lib-dynload
/usr/lib/python3.13/site-packages
```

DISTRIBUTION

why

- share and distribute code
- maintain versions
- map dependency

standalone

- normal python file
- easy to integrate

package

- multiple python files in directory
- geneally include `__init__.py`

pacman

	package manager	pip
shipped with	usually os	independent
installation	central db	depends
scope	full system	bundled list
format	os dependent	any

PIP **I**nstalls **P**ackages by Ian Bicking

history

history

- 1998 distutil for Python 1.6

history

- 1998 distutil for Python 1.6
- 2002 **Py**thon **P**ackage **I**ndex by Richard Jones

history

- 1998 distutil for Python 1.6
- 2002 **Py**thon **P**ackage **I**ndex by Richard Jones
- 2007 **v**irtual**env** by Ian Bicking

history

- 1998 distutil for Python 1.6
- 2002 **Py**thon **P**ackage **I**ndex by Richard Jones
- 2007 **v**irtual**env** by Ian Bicking
- 2008 **P**IP **I**nstalls **P**ackages by Ian Bicking

history

- 1998 distutil for Python 1.6
- 2002 **Py**thon **P**ackage **I**ndex by Richard Jones
- 2007 **v**irtual**env** by Ian Bicking
- 2008 **P**IP **I**nstalls **P**ackages by Ian Bicking
- 2009 **Py**thon **P**ackage **M**anager, ActiveState

history

- 1998 distutil for Python 1.6
- 2002 **Py**thon **P**ackage **I**ndex by Richard Jones
- 2007 **v**irtual**env** by Ian Bicking
- 2008 **P**IP **I**nstalls **P**ackages by Ian Bicking
- 2009 **Py**thon **P**ackage **M**anager, ActiveState
- 2011 **Py**thon **P**ackaging **A**uthority

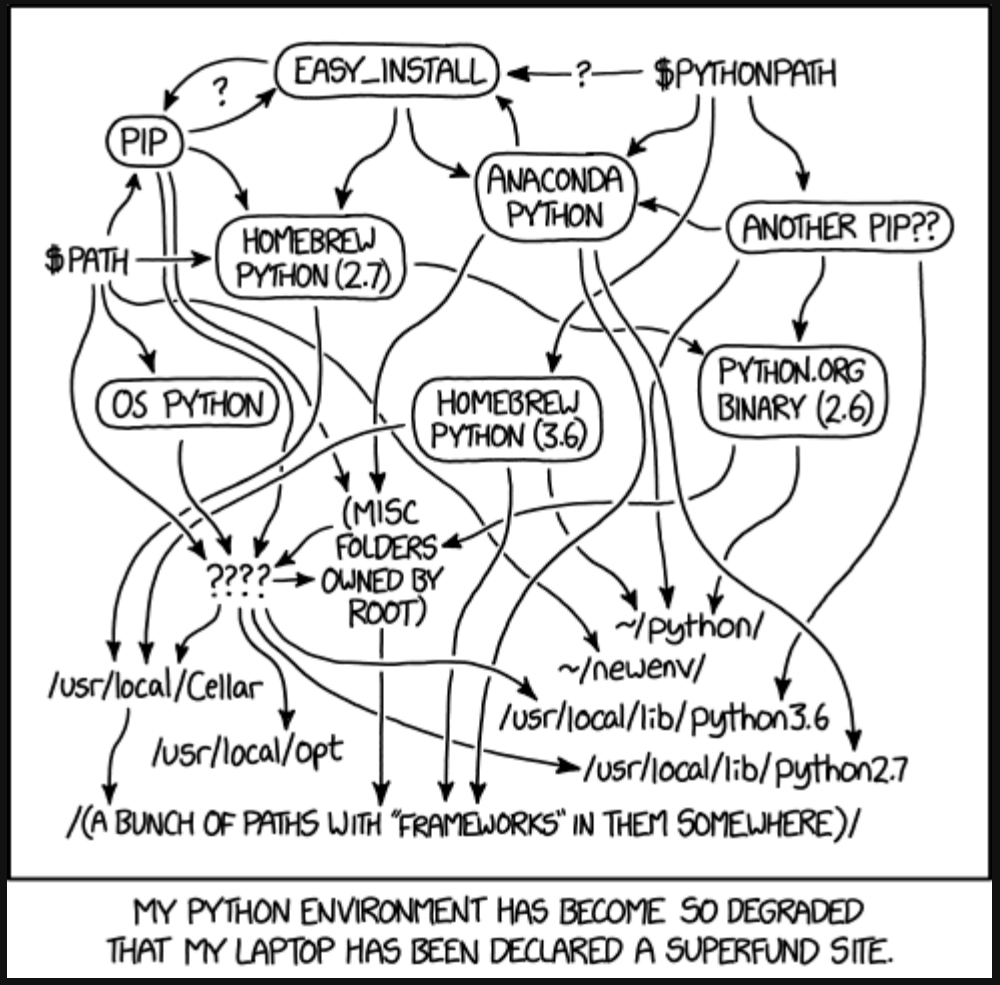
history

- 1998 distutil for Python 1.6
- 2002 **Py**thon **P**ackage **I**ndex by Richard Jones
- 2007 **v**irtual**env** by Ian Bicking
- 2008 **P**IP **I**nstalls **P**ackages by Ian Bicking
- 2009 **Py**thon **P**ackage **M**anager, ActiveState
- 2011 **Py**thon **P**ackaging **A**uthority
- 2012 Ana**conda** distribution

alternatives

distribution	package manager
PyPA	pip/buildout/enscons
Anaconda	conda/miniconda
ActiveState	Python Package Manager (PyPM)

VIRTUAL



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

why

- isolation: multiple version
- permission: no administrator rights
- organise: prevent package clutter

create

```
python -m venv venv  # 2nd venv is just a name
tree -L 2 venv
```

```
venv
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── Activate.ps1
│   ├── pip
│   ├── pip3
│   ├── pip3.12
│   ├── python -> /usr/bin/python
│   ├── python3 -> python
│   └── python3.12 -> python
├── include
│   └── python3.12
├── lib
│   └── python3.12
├── lib64 -> lib
└── pyvenv.cfg

7 directories, 11 files
```

activate

`.\venv\Scripts\activate` in windows

```
which python
source ./venv/bin/activate
which python
deactivate
which python
```

```
/usr/bin/python
/tmp/venv/bin/python
/usr/bin/python
```

pip list

```
source ./venv/bin/activate
python --version
pip list
```

```
Python 3.12.7
Package Version
------- -------
pip     24.2
```

pip install

```
./venv/bin/pip install numpy
./venv/bin/pip list
```

```
Collecting numpy
  Using cached numpy-2.1.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_
Using cached numpy-2.1.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x
Installing collected packages: numpy
Successfully installed numpy-2.1.3
Package Version
------- -------
numpy   2.1.3
pip     24.2
```

pip freeze

```
./venv/bin/pip freeze
./venv/bin/pip freeze > requirements.txt
```

```
numpy==2.1.3
```

```
# installing back
pip install -r requirements.txt
pip uninstall -r requirements.txt
```

PIP Installs Packages

pypi

python™
Package
Index

setup

pip is pre-installed Python >=3.4 in **windows**

```
apt install python3-pip    # debian/ubuntu
dnf install python3-wheel  # fedora >=22
apk add py-pip             # alpine
pacman -S python-pip       # archlinux

zypper install python3-pip # suse
yum install python3-pip    # centos/redhat/ibm
```

operations

```
# install from PyPi
pip install SomePackage

pip install --upgrade SomePackage

pip show --files SomePackage

pip uninstall SomePackage
```

version lock

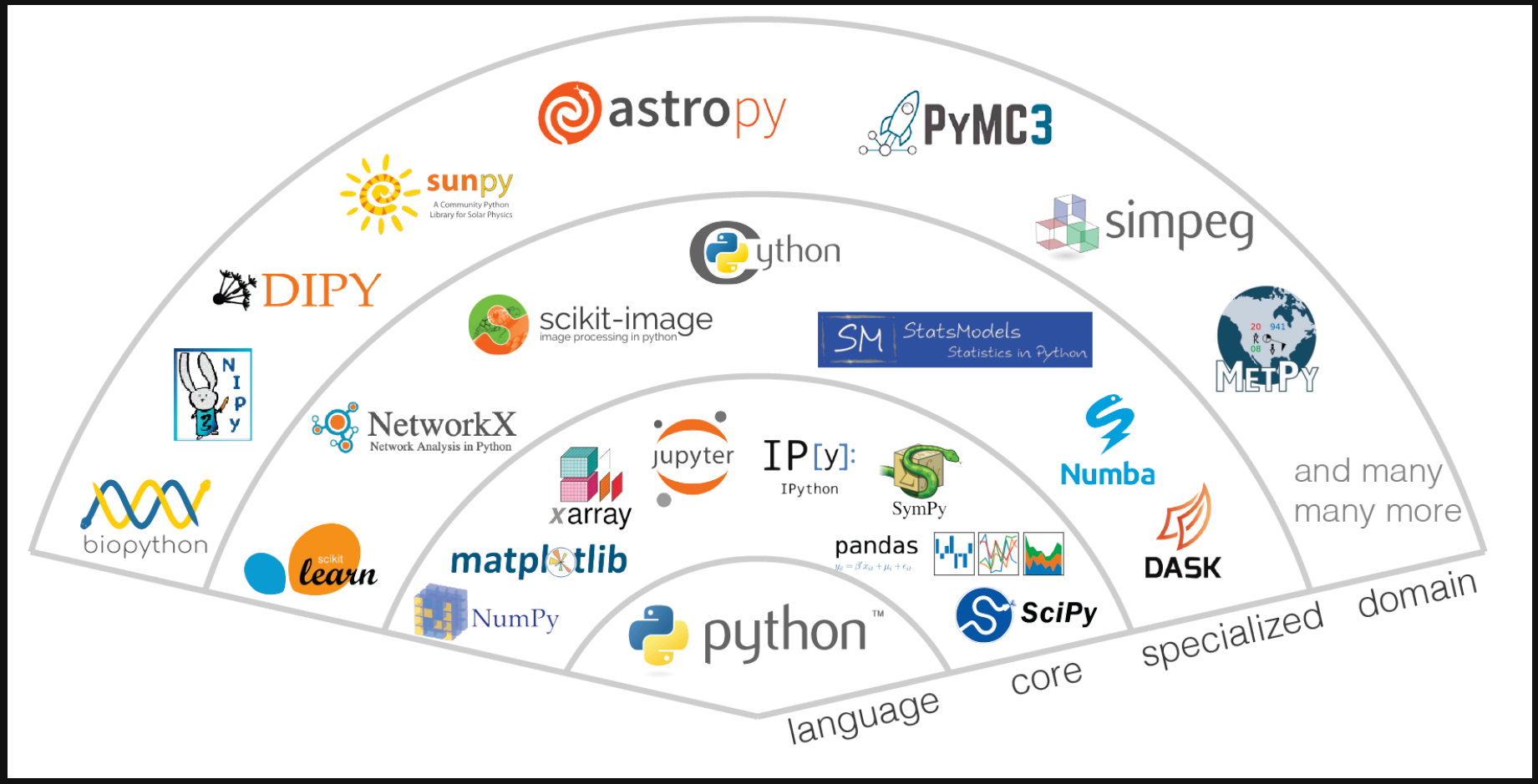
```
# version
pip install 'SomeProject==1.4.2'
pip install 'SomeProject>=1,<2'

# install downloaded file
pip install SomePackage-1.0-py2.py3-none-any.whl
```

update

```
pip install numpy --upgrade
```

ECOSYSTEM


astropy
PyMC3
sunpy
A Community Python Library for Solar Physics
simpeg
Cython
DIPY
scikit-image
image processing in python
SM
StatsModels
Statistics in Python
METPY
NIPY
NetworkX
Network Analysis in Python
jupyter
IP[y]:
IPython
SymPy
Numba
xarray
biopython
scikit learn
matplotlib
pandas
DASK
and many many more
NumPy
python
SciPy
language
core
specialized
domain

data exploration and transformation

- numpy
- pandas
- jupyter
- intake
- dask

visualization

- matplotlib: make hard things possible
- seaborn: statistical interface for matplotlib
- bokeh: interactive visualization for web
- Plotly: interactive visualization for web
- bokeh
- holoviews

Statistics

- Statsmodels: classes and functions for the estimation of statistical models, conducting statistical tests, and statistical data exploration
- Scikits: add-on packages for SciPy, hosted and developed separately and independently from the main SciPy distribution, providing more specialized functionality in a large number of topic areas

AI/ML

- scikit-learn: many different machine learning methods in Python
- TensorFlow: Deep Learning in Python (an end-to-end open source platform for machine learning)
- PyTorch: open source machine learning framework that accelerates the path from research prototyping to production deployment
- Keras: high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano
- Caffe: deep learning framework made with expression, speed, and modularity in mind

NLP

- nltk: platform for building Python programs to work with human language data
- spacy: industrial-strength natural language processing in Python
- textblob: Python library for processing textual data
- Gensim
- Transformers
- spaCy

web

- flask
- django