



Python in Space Physics

Snakes on a Spaceship



Angeline G. Burrell

U.S. Naval Research Laboratory, Space Science Division
AGB is supported by the Chief of Naval Research

Magnetosphere Online Seminar Series
5 October 2020

Outline



- Python in General
- Python in Space Physics, in Particular
- Responsible programming practices
- Resources

Python in General



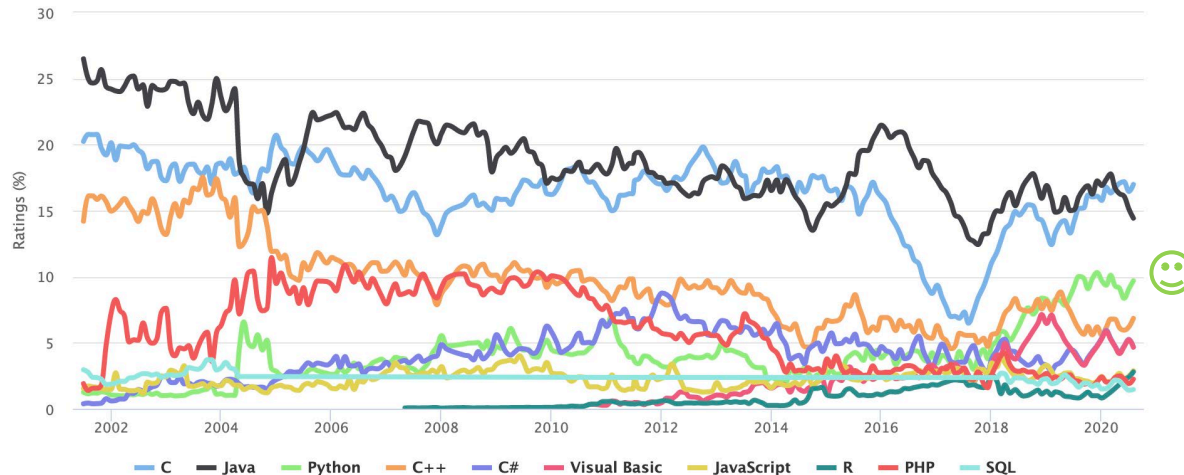
- Why Python?
- How to get started
- Going beyond “Hello World”

Python is ...

- ... a multi-paradigm (procedural, object oriented, etc.) general purpose language.
- ... named after a Flying Circus.
- ... relatively young: version 1.0 in 1994; version 2 in 2000 (IDL->1977; MatLab->1984).
- ... extremely popular and transferable to different disciplines and industries.
- ... Open source and **FREE!**

TIOBE Programming Community Index

Source: www.tiobe.com



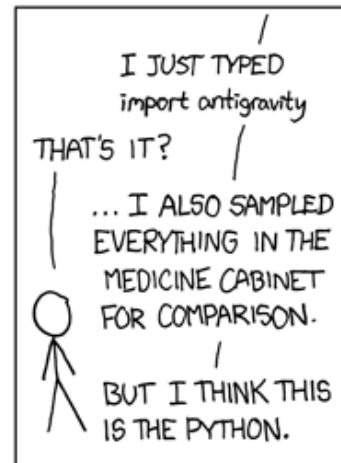
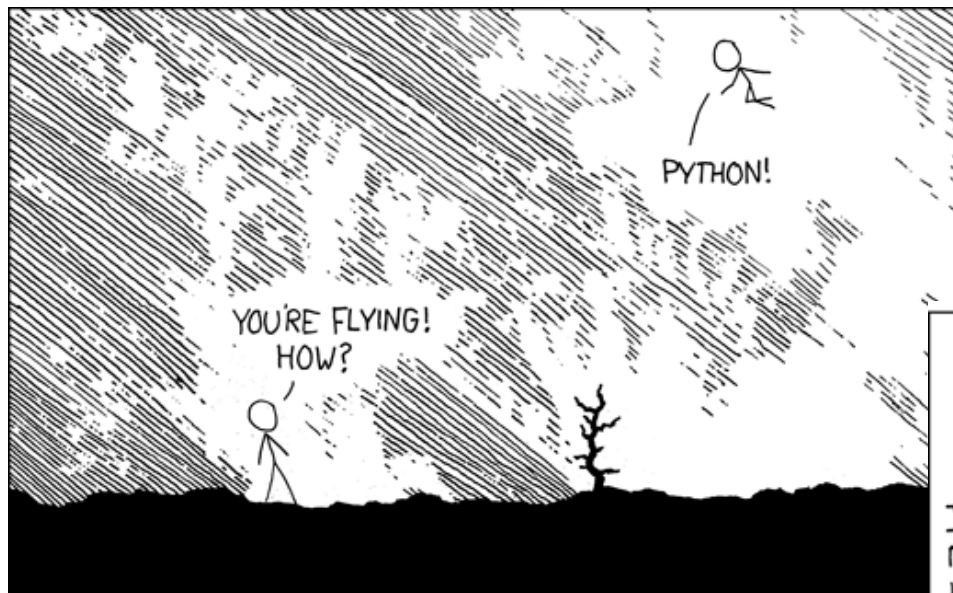
TIOBE Accessed 08/2020

Language	Rank
Python	3
MatLab	16
IDL	<50, ≥100

Current version: 3.8

Python is Fun

<https://xkcd.com/353/>
(slightly edited for Python 3 syntax)



Python is Useful

General Applications

- Powerful scripting rivaling Perl, Bash, etc.
- Ubiquitous across platforms (Win, *nix, OSX, and many more)
- Different IDEs and interactive shells accommodate different development habits
- Emphasizes clarity of source code
- Encourages good programming practices
- Includes most commonly used tools (epoch time, matrix operations, plotting, and more) “out of the box”

Scientific Applications

- Combines scripting with numerics and visualization
- Extensible with C, C++, and Fortran
- Large library of tested, citable scientific routines:
 - Numpy (array algebra)
 - SciPy (common scientific functions)
 - Matplotlib (visualization tools based on MatLab)
- Modules available to read common file types (ascii, HDF, CDF, netCDF, idl .sav, .m, etc.)

Python is Obtainable

Unix/Linux

1. Already installed!
2. Install a new version from source
3. Use a distribution like Anaconda
4. Use a package manager

Mac OSX

1. Comes with python... but do not use!
2. Install a new version from source
3. Use a distribution like Anaconda
4. Use a package manager

Windows

1. Install a new version from source
2. Use a distribution like Anaconda

Use your package manager, PyPi, or Anaconda to get new modules
Most scientists will want Python, Numpy, SciPy, Matplotlib, and iPython (a scripting shell)

Beyond “Hello World”

```
$ ipython
```

```
In [1]: import numpy as np
```

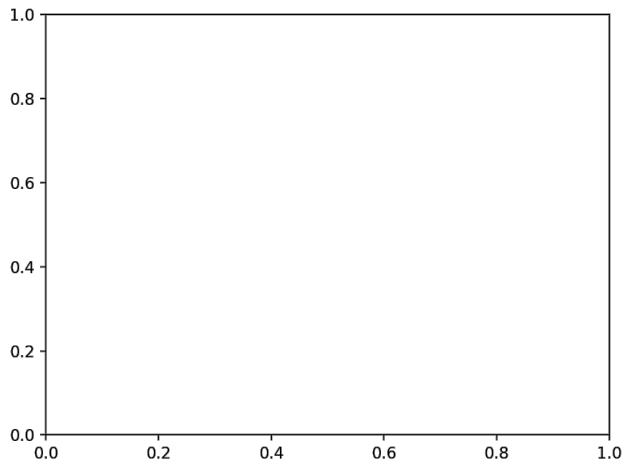
```
In [2]: import matplotlib as mpl
```

```
In [3]: import matplotlib.pyplot as plt
```

```
In [4]: plt.ion()
```

```
In [5]: fig = plt.figure()
```

```
In [6]: ax = fig.add_subplot(1,1,1)
```



- **ipython** begins interactive shell
- **import** command loads modules or python files into interactive namespace
- **ion()** command turns on interactive plotting

Beyond “Hello World”

\$ ipython

In [1]: import numpy as np

In [2]: import matplotlib as mpl

In [3]: import matplotlib.pyplot as plt

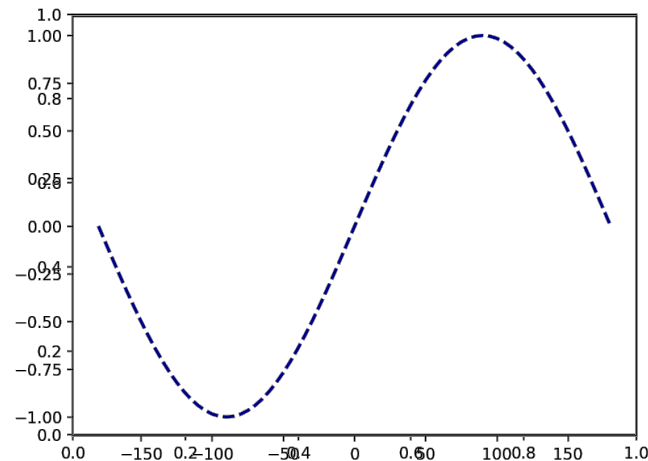
In [4]: plt.ion()

In [5]: fig = plt.figure()

In [6]: ax = fig.add_subplot(1,1,1)

In [7]: theta = np.arange(-180, 181, 5.0)

In [8]: ax.plot(theta, np.sin(np.radians(theta)), "--", color="navy", lw=2)



- **ipython** begins interactive shell
- **import** command loads modules or python files into interactive namespace
- **ion()** command turns on interactive plotting

Beyond “Hello World”

\$ ipython

In [1]: import numpy as np

In [2]: import matplotlib as mpl

In [3]: import matplotlib.pyplot as plt

In [4]: plt.ion()

In [5]: fig = plt.figure()

In [6]: ax = fig.add_subplot(1,1,1)

In [7]: theta = np.arange(-180, 181, 5.0)

In [8]: ax.plot(theta, np.sin(np.radians(theta)), "--", color="navy", lw=2)

In [9]: ax.set_xlim(theta.min(), theta.max())

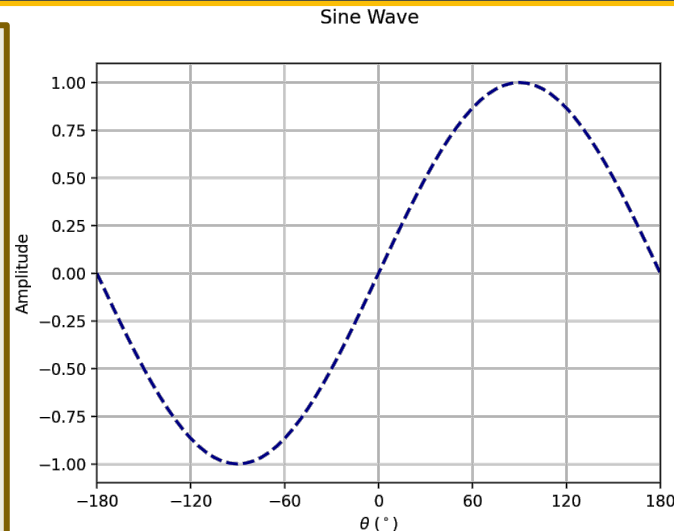
In [10]: ax.xaxis.set_major_locator(mpl.ticker.MultipleLocator(60))

In [11]: ax.set_xlabel(r"\$\theta\$ (\$^\circ\$")

In [12]: ax.set_ylabel("Amplitude")

In [13]: ax.grid()

In [14]: fig.suptitle("Sine Wave")



- **ipython** begins interactive shell
- **import** command loads modules or python files into interactive namespace
- **ion()** command turns on interactive plotting
- LaTeX style commands yield publication-quality labels

Python in Space Physics, in Particular



- Many Python space physics packages
- A closer look at:
 - Multipurpose
 - Observational data access
 - Modelled data tools
 - Coordinates
 - Orbits and ephemeris
 - Data analysis and file routines
 - Scientific workflow

Python in Space Physics

Burrell et al. (2018) doi:[10.1029/2018JA025877](https://doi.org/10.1029/2018JA025877)

Data Access and Analysis

Modeled Data Access and Analysis

- analysator
- OvationPyme
- pyAMPS
- pyForecastTools
- pyglow

Observational Data Access and Analysis

- DaViTpy/pyDARN
- digital_rf
- GeoData
- HelioPy
- MadrigalWeb
- pysat

Data Analysis and File Routines

- CDFlib
- pyLTR
- pysatCDF

Coordinates

- AACGMV2
- Apexpy
- OCBpy
- OMMBV

Multipurpose

- AstroPy
- geospacepy
- SpacePy
- SunPy
- PlasmaPy

Orbits

- SpiceyPy
- PyEphem
- SGP4
- skyfield and jplephem

Heliophysics Field Key

- Sun/Solar Wind
- Magnetosphere
- Ionosphere/Thermosphere/Mesosphere
- Other



Even more

- AMGeO
- HIME
- pySPEDAS
- ReesAurora
- RESEN
- ssj_auroral_boundaries
- Tsyganenko
- Viresclient

Multipurpose: SpacePy



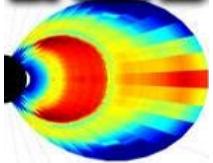
- J. Niehof, B. Larsen, S. Morley, D. Welling
- Documentation: spacepy.github.io
- Goals:
 - Quickly obtain data
 - Create publication quality plots
 - Run common empirical models
 - Change coordinates easily
- Works on Mac, Windows, and Linux/Unix
- Recent presentation given to PyHC: drive.google.com/drive/u/0/folders/1i9_bnmMOHadUWwFo92AkIMbHTe_ylzocx

Code breakdown

- Location and Time
 - **time**: datetime, GPS, Julian, Georgian ordinal
 - **coordinates**: geodetic, geographic, GSM, GSE, solar magnetic, geomagnetic, spherical, and Cartesian
 - **irbempy**: field-line tracing from different magnetic field models
- File handling
 - **ae9ap9**: read AE6/AP9 data files
 - **pyCDF**: read and write NASA CDF files
 - **omni**: handle different types of OMNI data
 - **datamodel**: handles HDF5, netCDF, and JSON ASCII
- **datamodel**: common class for all loaded data, simplifying analysis
- Model tools
 - **data_assimilation**: Ensemble-based data assimilation subroutines for the Radiation Belt Model
 - **empiricals**: includes empirical models for the maximum L-shell, plasmopause, solar proton spectra, pitch angle, and more
 - **PyBats**: BATS-R-US and SWMF output
 - **radbelt**: radiation belt diffusion codes
- Analysis and visualization tools
 - **plot**: specialized plotting functions for common visualizations, including spectrograms
 - **PoPPy**: point process to assess statistical association between time series
 - **SeaPy**: superposed epoch analysis
 - **toolbox**: binning, searching, masking, and other tools

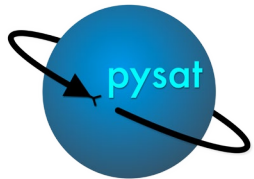
Observational Data Access

SPEDAS



- pySPEDAS
 - Contains data loading, data analysis and data plotting tools for various scientific data sets
 - Recent presentation from this seminar series
 - Covers 22 missions, many with multiple instruments
 - github.com/spedas/pyspedas

VirES



- MadrigalWeb
 - Tools for downloading data from the Madrigal database
 - GeoData provides friendlier interface
 - github.com/jsowoda/GeoData
 - cedar.openmadrigal.org/docs/name/rr_python.html

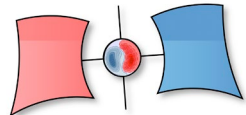
HelioPy



- VirESclient
 - Connects to the ESA VirES server to obtain data
 - Returns data in pandas or xarray objects
 - github.com/ESA-VirES/VirES-Python-Client
- pyDARNio
 - Library for loading SuperDARN data
 - Works with pyDARN, a visualization tool
 - github.com/SuperDARN/pyDARNio

- pysat
 - An interface for downloading, loading, cleaning, managing, processing, and analyzing data
 - Current version (2.2+) is self contained
 - New version (3.0+) will be streamlined for operational use by providing a framework
 - Breaks out instruments into modules by source or purpose, to minimize dependencies
 - Modules include tools for mission design, seasonal analysis, space weather drivers, Madrigal data, NASA data, and CDAAC data
 - github.com/pysat/pysat
- HelioPy
 - provides a set of generic tools to work with 1D *in situ* time-series measurements of space plasmas
 - Provides access to Solar and Magnetospheric observations
 - Wraps the SPICE library and has further solar physics tools
 - github.com/heliopython/heliopy

Modelled Data Tools



- AMGeo

- AMGeo implements data assimilation analysis within the Assimilative Mapping of Ionospheric Electrodynamics (AMIE)
- Accepts SuperDARN and SuperMAG
- amgeo.colorado.edu



- Analysator

- An analysis tool for Vlasiator, a hybrid-Vlasov model that resolve non-MHD Space Weather processes
- github.com/fmihpc/analysator

- HIME

- The High-latitude Input for Meso-scale electrodynamics (HIME) determines the upper boundary conditions (drivers) for global ionosphere-thermosphere models using ISR and electric field measurements
- github.com/dcszoturk/hime

- pyForecastTools

- Provides model validation and forecast verification tools

- Works seamlessly with SpacePy
- Also contains visualization tools
- github.com/drsteve/PyForecastTools

- pyAMPS

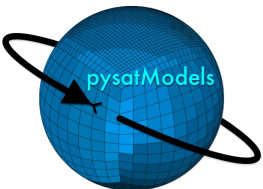
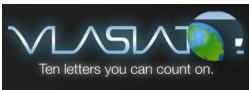
- An interface for the Average Magnetic field and Polar current System (AMPS) model.
- github.com/klaundal/pyAMPS

- pysatModels

- Provides an interface for (potentially) downloading, loading, managing, processing, and analyzing model data
- Contains tools useful for model-measurement and model-model comparisons
- Uses pyForecastTools in validation tools
- github.com/pysat/pysatModels

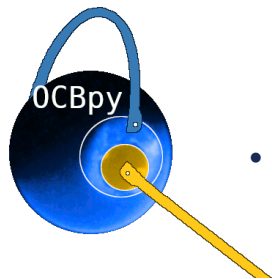
- ReesAurora

- A python wrapper for the Rees-Sergienko-Ivanov model of excitation rates based on AIDA_Tools
- github.com/space-physics/reesaurora



Coordinates

- apexpy
 - A Python wrapper for the Apex Fortran library by Emmert et al. [2010]
 - Converts between geodetic, modified apex, and quasi-dipole coordinates
 - Calculates modified apex and quasi-dipole base vectors
 - Calculates magnetic local time
 - github.com/aburrell/apexpy
- AACMGV2
 - The official Python wrapper for the AACGM-v2 C library, which allows converting between geographic or geodetic and altitude-adjusted corrected magnetic coordinates.
 - github.com/aburrell/aacgm2
- OCBpy
 - Convert between standard magnetic or geographic coordinates and a magnetic coordinate system that adjusts latitude and local time relative to the Open-Closed field line Boundary (OCB)
- OOMBV
 - github.com/aburrell/ocbpy
 - Orthogonal Multipole Magnetic Basis Vectors
 - The first to remain orthogonal for multipole magnetic fields as well as when including a geodetic reference surface (Earth)
 - github.com/rstoneback/OMMBV
- Tsyganenko
 - A python wrapper for the Tsyganenko geomagnetic field Fortran routines
 - github.com/johncoxon/tsyganenko



Orbit and Ephemeris

- JPLEphem

- A package for users of Jet Propulsion Laboratory (JPL) ephemeris
 - Depends on PyEphem
 - Recommends Skyfield if working in non-cartesian coordinate systems

- pypi.org/project/jplephem

- PyEphem

- Legacy package that enables the computation of planet, comet, asteroid, and Earth satellite positions
 - Wraps the libastro C library
 - Maintained, but recommends using Skyfield

- rhodesmill.org/pyephem/index.html

- SGP4

- Computes the position and velocity of an earth-orbiting satellite, given the satellite's TLE orbital elements

- Uses either pure Python or the the official C++ version of SGP4
- Works in the “True Equator Mean Equinox” reference frame

- github.com/brandon-rhodes/python-sgp4

- Skyfield

- Computes positions for the stars, planets, and satellites in orbit around the Earth
 - Supports several coordinate systems
 - Uses pure Python

- rhodesmill.org/skyfield

- SpiceyPy

- Provides tools for planning and interpreting L1 scientific observations from space-borne instruments
 - Wraps the NAIF C SPICE Toolkit (N66)
- github.com/AndrewAnnex/SpiceyPy



Data Analysis and File Routines

- CDFLib
 - Reads NASA CDF files and does not depend on the NASA library
 - github.com/MAVENSDC/cdflib
- pysatCDF
 - Reads NASA CDF files by wrapping the NASA Fortran library
 - github.com/pysat/pysatCDF
- h5py
 - Reads and writes HDF5 files
 - www.h5py.org/
- netCDF4-python
 - Reads and writes netCDF files by wrapping the netCDF C library
 - github.com/Unidata/netcdf4-python
- pyLTR
 - A learning-to-rank (LTR) toolkit for information retrieval and data mining
 - LTR is a supervised machine learning method that trains a model to rank lists of data points rather than score single data points
 - github.com/jma127/pyltr
- ssj_auroral_boundaries
 - Identify auroral boundaries from the Defense Meteorology Satellite Program (DMSP) electron precipitation
 - github.com/lkilcommons/ssj_auroral_boundary

Scientific Workflow: RESEN

RESEN

- REproducible Software ENvironment is designed to improve reproducibility and collaboration
 - Provides pre-installed community software
 - Packages data analysis code and results in an isolated docker container
 - Each container can be shared, stored, and made citable using Zenodo
 - Can be run locally or online
- Created and maintained by the InGeo team for the geospace community
- Recent presentation at CEDAR:
http://cedarweb.vsp.ucar.edu/wiki/images/f/f6/Snakes_on_spaceship_2020_resen.pdf
- ingeo.datatransport.org/home/resen

Preloaded Packages

- | | |
|-----------------------|-----------------------|
| 1. aacgm2 | 13. numpy |
| 2. apexpy | 14. pandas |
| 3. basemap | 15. pyglow |
| 4. bokeh | 16. pymongo |
| 5. cartopy | 17. scipy |
| 6. davitpy | 18. sciunit2 |
| 7. h5py | 19. spg4 |
| 8. ipython | 20. spacepy |
| 9. madrigalweb | 21. sqlalchemy |
| 10. mangopy | 22. sympy |
| 11. matplotlib | 23. tables |
| 12. netcdf4 | |

Responsible Programming



- Best practices
- Attribution
- Accessibility

Best Practices

- Support the current language version (3.8)
- Use the style of the language you are programming in
 - PEP 8 describes the pythonic style
 - PEP 257 describes the docstring conventions, though many projects follow the numpydoc standard
 - For collaborative projects, conform to the project's style and design standards
 - refactoring.guru/design-patterns/python
- Use descriptive variables
- Comment your code!
 - For Python, use both docstrings and in-line comments
- Avoid duplication
- Provide documentation

```
# d is an array of floats or ints
# n is the length of d
def average(d, n):
    s = 0.0 # float, sum of d

    if(n<=0)
        return None
    if(n==1)
        return(d[0])

    for j in range(n):
        s += d[j]
    s /= n
    return(s)
```

Best Practices

- Support the current language version (3.8)
- Use the style of the language you are programming in
 - PEP 8 describes the pythonic style
 - PEP 257 describes the docstring conventions, though many projects follow the numpydoc standard
 - For collaborative projects, conform to the project's style and design standards
 - refactoring.guru/design-patterns/python
- Use descriptive variables
- Comment your code!
 - For Python, use both docstrings and in-line comments
- Avoid duplication
- Provide documentation

```
import numpy as np

def average(d):
    """ calculate the average

    Parameters
    -----
    d : array-like
        List or array of value to include in average

    Returns
    -----
    s : float
        Average of values in data or NaN for empty input

    """
    s = np.nan

    if len(d) == 1:
        s = d[0]
    else:
        s = np.sum(d) / len(d)

    return s
```

Best Practices

- Support the current language version (3.8)
- Use the style of the language you are programming in
 - PEP 8 describes the pythonic style
 - PEP 257 describes the docstring conventions, though many projects follow the numpydoc standard
 - For collaborative projects, conform to the project's style and design standards
 - refactoring.guru/design-patterns/python
- Use descriptive variables
- Comment your code!
 - For Python, use both docstrings and in-line comments
- Avoid duplication
- Provide documentation

```
import numpy as np

def average(data):
    """ calculate the average

    Parameters
    -----
    data : array-like
        List or array of value to include in average

    Returns
    -----
    ave : float
        Average of values in data or NaN for empty input

    """
    ave = np.nan

    if len(data) == 1:
        ave = data[0]
    else:
        ave = np.sum(data) / len(data)

    return ave
```

Best Practices

- Support the current language version (3.8)
- Use the style of the language you are programming in
 - PEP 8 describes the pythonic style
 - PEP 257 describes the docstring conventions, though many projects follow the numpydoc standard
 - For collaborative projects, conform to the project's style and design standards
 - refactoring.guru/design-patterns/python
- Use descriptive variables
- Comment your code!
 - For Python, use both docstrings and in-line comments
- Avoid duplication
- Provide documentation

```
import numpy as np

def average(data):
    """ calculate the average

    Parameters
    -----
    data : array-like
        List or array of value to include in average

    Returns
    -----
    ave : float
        Average of values in data or NaN for empty input

    """
    # If unable to calculate an average, return NaN for easy masking
    ave = np.nan

    # Average is sum / num_points, which is just the data value for
    # one data point
    if len(data) == 1:
        ave = data[0]
    else:
        ave = np.sum(data) / len(data)

    return ave
```


Best Practices

- Support the current language version (3.8)
- Use the style of the language you are programming in
 - PEP 8 describes the pythonic style
 - PEP 257 describes the docstring conventions, though many projects follow the numpydoc standard
 - For collaborative projects, conform to the project's style and design standards
 - refactoring.guru/design-patterns/python
- Use descriptive variables
- Comment your code!
 - For Python, use both docstrings and in-line comments
- Avoid duplication
- Provide documentation

```
import numpy as np
```

```
# Get average of all data
```

```
ave = np.mean(data)
```

```
# Get average of all good data (exclude NaN)
```

```
good_ave = np.nanmean(data)
```

Best Practices


- Support the current language version (3.8)
- Use the style of the language you are programming in
 - PEP 8 describes the pythonic style
 - PEP 257 describes the docstring conventions, though many projects follow the numpydoc standard
 - For collaborative projects, conform to the project's style and design standards
 - refactoring.guru/design-patterns/python
- Use descriptive variables
- Comment your code!
 - For Python, use both docstrings and in-line comments
- Avoid duplication
- Provide documentation



- Sphinx is a Python package that helps generate and maintain software documentation
 - Uses reStructuredText
 - Has options to create several output formats: HTML, LaTeX, ePub, Texinfo, manual pages, and plain text
 - Can generate basic documentation from docstrings
 - Extensions allow documentation to be tested (e.g., identify broken links)
 - Works well with document hosting tools such as readthedocs
- readthedocs.org
- docutils.sourceforge.io/rst.html
- www.sphinx-doc.org

Attribution

- Scientific software and algorithms should be cited
 - Provides methodological clarity
 - Credits traditionally underappreciated areas of our field
- To cite a software project:
 - Follow the guidelines provided by the project
 - If no guidelines are given, use a webpage-style citation
- To improve citability:
 - Repositories like Zenodo allow versioned DOIs
 - Journals allow software and technique publications
 - Journal of Geophysical Research
 - Journal of Open Source Software


scipy.org/citing.html

SciPy.org

Citing packages in the SciPy ecosystem

A number of articles related to scientific computing with Python have appeared; a selection related to some of the core toolstack are listed below. See also the [May 2007](#) and [March 2011](#) editions of the journal *Computing in Science & Engineering*, which focuses on scientific computing with Python.

SciPy (the library)

There is now a journal article available for citing usage of SciPy:

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C.J. Carey, Iihan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, in press.

Here's an example of a BibTeX entry:

```

@ARTICLE{2020SciPy-NMeth,
  author = {(Virtanen), Pauli and (Gommers), Ralf and (Oliphant),
    Travis E. and (Haberland), Matt and (Reddy), Tyler and
    (Cournapeau), David and (Burovski), Evgeni and (Peterson), Pearu
    and (Weckesser), Warren and (Bright), Jonathan and (van der Walt),
    St{\'e}fan J. and (Brett), Matthew and (Wilson), Joshua and
    (Jarrod Millman), K. and (Mayorov), Nikolay and (Nelson), Andrew
    R.-J. and (Jones), Eric and (Kern), Robert and (Larson), Eric and
    (Carey), C.J. and (Polat), {\.{I}}lhan and (Feng), Yu and (Moore),
    Eric W. and (Vand erPlas), Jake and (Laxalde), Denis and
    (Perktold), Josef and (Cimrman), Robert and (Henriksen), Ian and
    (Quintero), E.-A. and (Harris), Charles R. and (Archibald), Anne M.
    and (Ribeiro), Ant{\'o}nio H. and (Pedregosa), Fabian and
    (van Mulbregt), Paul and (Contributors), SciPy 1. 0),
  title = "{SciPy 1.0: Fundamental Algorithms for Scientific
    Computing in Python}",
  journal = {Nature Methods},
  year = "2020",
  volume={17},
  pages={261--272},
  adsurl = {https://rdcu.be/b0EWb},
  doi = {https://doi.org/10.1038/s41592-019-0686-2},
}

```

This supersedes the preprint in [arXiv:1907.10121](#)

For any specific algorithm, also consider citing the original author's paper (this can often be found under the "References" section of the docstring).

About SciPy
Getting started
Documentation
Install
Bug reports
Codes of Conduct
SciPy conferences
Topical software

Citing
Cookbook
Blogs
NumFOCUS

CORE PACKAGES:
NumPy
SciPy library
Matplotlib
IPython
SymPy
pandas

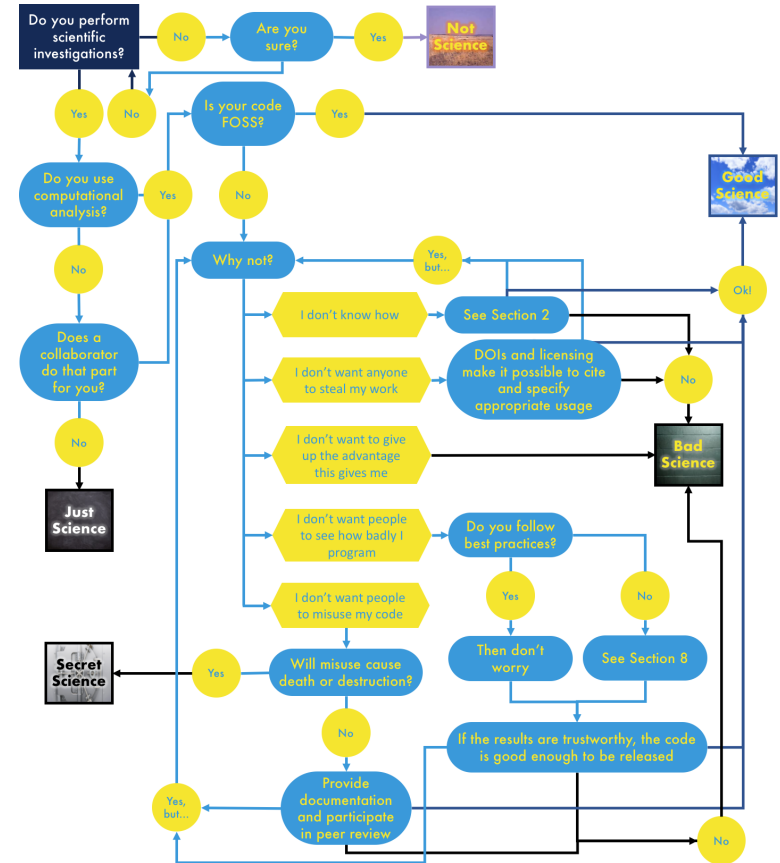
Table of Contents

- Citing packages in the SciPy ecosystem
 - SciPy (the library)
 - Scientific computing in Python
 - NumPy
 - IPython
 - Matplotlib
 - Cython
 - pandas
 - scikit-learn
 - scikit-image
 - F2PY
 - SymPy

Search

Accessibility

- Publish your analysis code
 - Repositories such as GitHub, GitLab, and Bitbucket all:
 - Use git for version control
 - Allow public and private repositories
 - Have project management tools
 - Repositories like Zenodo and FigShare accept notebooks, data sets, and docker containers (like RESEN)



Accessibility

- Choose the right license for your code
 - Morin et al., (2012). A quick guide to software licensing for the scientist-programmer. *PLOS Computational Biology*, 8(7), e1002598
 - choosealicense.com
 - Common Free and Open Source Software (FOSS) Licenses:
 - Permissive: MIT, BSD
 - Copyleft: GPL, LGPL
 - Talk to your institution
- Respect the work you are building upon
 - Talk with algorithm developers and determine best method of acknowledgement
 - Follow copyright rules
 - When contributing to a repository, follow the package guidelines and code style

Resources

www.python.org	Source, documentation, other resources.
www.python.org/dev/peps/pep-0008	Style guide (suggested coding conventions)
Dive Into Python 3 (Mark Pilgrim)	Open-source introduction to Python (online and in print)
matplotlib.org	Matplotlib documentation
docs.scipy.org/doc	Numpy/SciPy documentation
pypi.org	Repository for most Python packages
heliopython.org	Collection of Space Physics packages
cedarweb.vsp.ucar.edu/wiki/index.p hp/Community:Software	CEDAR Wiki page for community software