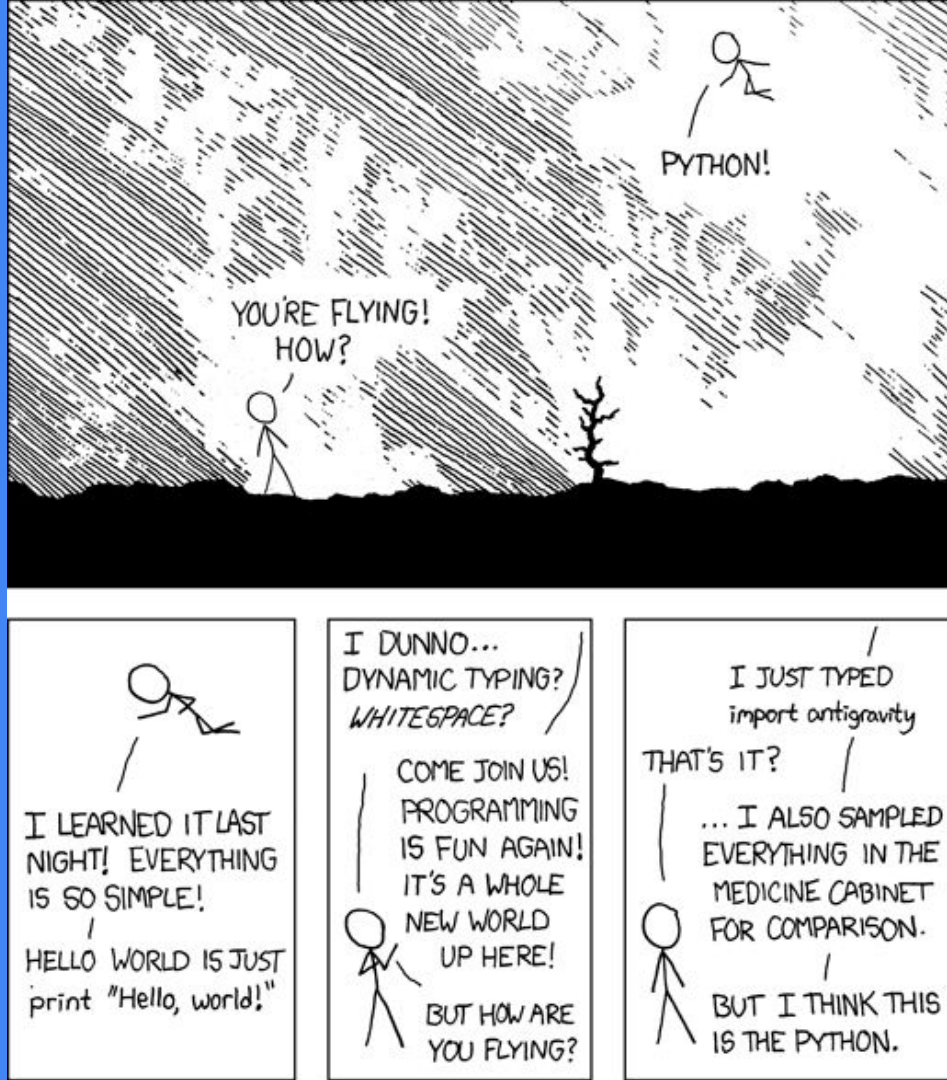
Python

How I Learned to Stop Worrying and Love to Code



IAASARS/NOA - 02/13/2019

Program

1. Introduction
2. Good practices in Python
3. Scientific programming
4. Python for astronomers
    a. Astropy: Main package
    b. Astropy: Associated packages
    c. Others useful packages (astLib, Sherpa…)
    d. Machine learning tools

Introduction

Why Programming?

It makes your life easier

Why Programming?

Reproducibility and Sharing

Why Programming?

Big Data Era

Why Programming?

Useful, and marketable, outside astronomy

Why Python?

Easy but powerful language

Why Python?

Python is free

Why Python?

Easy to bind with pre-existent libraries in C, C++ and Fortran

Why Python?

Large ecosystem of 3rd party modules and packages

Why Python?

We are scientist,
no software developers

Resources for beginners

- Python4astronomers
  Workshop: https://python4astronomers.github.io
- Python for Astronomers: An Introduction to Scientific Computing
  Book and practical examples: https://prappleizer.github.io
- A Primer on Scientific Programming with Python
  Book: https://www.springer.com/gp/book/9783662498866
- Statistics, Data Mining, and Machine Learning in Astronomy
  Book and on-line tutorial: http://www.astroml.org

Good Practices in Python

General Advices

- Write clean and easy to read code:
  - Good naming conventions
  - Modular (functions and classes)
  - Useful comments

General Advices

- Don’t reinvent the wheel:
  - Use modules and packages
  - Use Google!

The Zen of Python

Long time Pythoneer Tim Peters succinctly channels the BDFL's guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.

Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
*Although never is often better than *right* now.*
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Version control systems

Keeping a history of changes in your code.

- Git
- Mercurial
- SVN

Virtual environments

- A way of isolating your programming environment from your system.
- Different versions of Python (e.g. 2.4, 2.6, 3.6, 3.7) and packages in the same system.

Virtual environments: Anaconda

```
> wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
> bash Miniconda3-latest-Linux-x86_64.sh
> conda create -n python_workshop python=3.7 scipy ipython matplotlib
> source activate python_workshop
> conda install jupyter astropy
> pip install mocpy pymoc
> source deactivate

> conda config --add channels conda-forge
```

https://conda.io/en/latest/miniconda.html

Virtual environments: venv

```
> sudo apt-get install python3-venv
> python3 -m venv python_workshop
> source python_workshop/bin/activate
> pip install scipy ipython matplotlib astropy jupyter pymoc mocpy
> deactivate
```

Integrated Development Environment (IDE)

- Increase your productivity.
- Avoid most common errors with syntax.
  - Spyder

    ```
    > conda install spyder
    ```
  - PyCharm

ipython and jupyter notebooks

- ipython:
  - Command shell for interactive computing.
  - More powerful and easy to use than the standard python shell.
- jupyter notebooks:
  - Using ipython through a web browser.
  - Very useful for early prototyping.
  - Allows easy sharing of small scripts, recipes, etc.