An introduction to Python

T. Verstraelen

Center for Molecular Modeling, Ghent University, Belgium

Toon.Verstraelen@UGent.be

Scripting versus compiled languages

Scripting languages (interpreted)

- Bash
- Python
- Perl
- Matlab / Octave
- Basic
- TCL

...

Faster development
Slower execution
More features
No compilation

(Byte-)Compiled languages

- C
- C++
- Fortran
- Cython
- Basic
- Pascal
- Java(script)

...

Slower development
Faster execution
Less features
Compilation

Brief history of Python

1994

- First release (1.0)
- Main developer: Guido Van Rossum (BDFL)
- Influences from lisp, ABC, C++, ...
- Small community (not popular)

2000

- Influential release (2.0)
- Mature language, easier to use
- Open community, widespread adoption
- Latest in this series 2.7.x (most popular version)

2008

- Ideological cleanup release (3.0)
- Remove redundancies: less features
- Fix design mistakes
- Not backward compatible
- Slow adoption

Useful Python links

- The website:
  http://www.python.org/
- Python Tutorial
  http://docs.python.org/tutorial/index.html
- Reference of standard libraries
  http://docs.python.org/library/index.html
- Extensive Q&A site
  http://stackoverflow.com/
- Python package database
  http://pypi.python.org/pypi

Useful Python links for computational sciences

- Basic numerical tools: Numpy, Scipy and Matplotlib
  http://www.scipy.org/
- Storage of numerical data in files: HDF5
  http://www.h5py.org/
- Scikit-learn: machine learning, data sciences
  http://scikit-learn.org/stable/
- Just-in-time compilation
  https://numba.pydata.org/
  http://www.deeplearning.net/software/theano/
- Combining Python and C++
  http://cython.org/
  https://github.com/pybind/pybind11
- Arbitrary precision and symbolic math
  http://mpmath.org/
  http://www.sympy.org/
- Automatic analytic differentiation
  https://github.com/HIPS/autograd
- Cheminformatics
  http://www.rdkit.org/
- A longer list:
  https://scipy.org/topical-software.html

The Python interpreter (1/2)

Python that comes with your Linux or OSX

- Open a terminal
- Start the `python` or `python3` program. You'll get the Python command line:

```
Python 2.7.13 (default, Jun 26 2017, 10:20:05)
[GCC 7.1.1 20170622 (Red Hat 7.1.1-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Enter code, line by line, e.g. `1 + 1`
- To exit the interpreter: type `exit()` or press CTRL+d.

In a conda environment (Windows, Linux, OSX)

- Linux, OSX: open a terminal and activate your conda environment

```
. <your_conda_directory>/bin/activate
```

- Windows: open the anaconda prompt
- Start the `python` program. You'll get the Python command line:

```
Python 2.7.13 |Continuum Analytics, Inc.| (default, Dec 20 2016, 23:09:15)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>>
```

The Python interpreter (2/2)

Linux on HPC (gengar and friends)

- Load a Python module

```
module load Python/3.6.1-intel-2017a
```

- Start the `python` program. You'll get the Python command line:

```
Python 3.6.1 (default, May  9 2017, 16:24:49)
[GCC Intel(R) C++ gcc 6.3 mode] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python scripts (Linux, OSX)

- Create a file, e.g. script.py
- First line (shebang): `#!/usr/bin/env python`
- Comments start with hash: `#`
- Add some lines of Python code

```
print("Hello world")
```

- Make executable (once) and run:

```
chmod +x script.py
./script.py
```

- Or execute directly:

```
python ./script.py
```

How to edit code?

- Integrated development environment (IDE):
  - Sublimetext https://www.sublimetext.com/
  - Pycharm https://www.jetbrains.com/pycharm/
  - Atom https://atom.io/
  - ...
- Text editors:
  - Notepad++ (Windows) https://notepad-plus-plus.org/
  - Gedit (Linux, Gnome)
  - Kate (Linux, KDE)
  - Vim (terminal, Linux, OSX)
  - Emacs (terminal, Linux, OSX)
  - Nano (terminal, Linux, OSX, too simple, don't use)
  - ...

Good practices: coding style

- Main purpose: **improve readability**
- The basics (taken from PEP8):
  - Use 4-space indentation, no tabs. (Configure your editor.)
  - Wrap lines (don't exceed 100 characters).
  - Separate blocks of code by empty line(s).
  - Put comments on a line of their own. Space after hash.

```
# This is a comment on the following line.
print(2)
```

  - Write docstrings.

```
def add(a, b):
    """Compute sum of a and b."""
    return a + b
```

  - Put spaces around operators and after commas, not around brackets: `a = f(1, 2) + g(3, 4)`
  - Class names: `CamelCase`
    Module-level constant names: `UPPER_CASE_WITH_UNDERSCORES`
    Variable, function and method names: `lower_case_with_underscores`

Good practices: use “linters” (1/2)

- Main purpose: **check for style issues and potential bugs**
- Popular choices: pylint, pycodestyle, pydocstyle
- Example of poor style `poor.py`

```
def add(a,b):
    '''Adds stuff.'''
    return a+b
    c = a+b
```

- Output of `pylint poor.py`

```
************* Module poor
C:  1, 0: Exactly one space required after comma
def add(a,b):
         ^ (bad-whitespace)
C:  1, 0: Missing module docstring (missing-docstring)
C:  1, 0: Invalid argument name "a" (invalid-name)
C:  1, 0: Invalid argument name "b" (invalid-name)
W:  4, 4: Unreachable code (unreachable)
C:  4, 4: Invalid variable name "c" (invalid-name)
W:  4, 4: Unused variable 'c' (unused-variable)

-----------------------------------
Your code has been rated at -13.33/10
```

Good practices: use “linters” (2/2)

- Main purpose: **check for style issues and potential bugs**
- Popular choices: pylint, pycodestyle, pydocstyle
- Example of poor style `poor.py`

```
def add(a,b):
    '''Adds stuff.'''
    return a+b
    c = a+b
```

- Output of `pycodestyle poor.py`

```
poor.py:1:10: E231 missing whitespace after ','
```

- Output of `pydocstyle poor.py`

```
poor.py:1 in public function `add`:
        D401: First line should be in imperative mood ('Add', not 'Adds')
poor.py:1 in public function `add`:
        D300: Use """triple double quotes""" (found '''-quotes)
```

Good practices: write unit tests

- Main purpose: **validate features of your code**
- Example: `adder.py`

```
def add(a, b):
    """Compute the sum of a and b."""
    return a + b


def test_add():
    assert add(1, 2) == 3
    assert add(2, -3) == -1
```

- Run the tests with nose: `nosetests -v adder.py`

```
adder.test_add ... ok

----------------------------------------------------------------------
Ran 1 test in 0.085s

OK
```

Good practices: reuse code

- Main purpose: **do not reinvent the wheel**
- Do not copy-paste someone else code (copyright and not sustainable)
- Install someone else's software instead:

```
# Conda, any OS
conda install <package_name>

# Python package index, any OS
pip install --user <package_name>

# Fedora Linux
sudo dnf install <package_name>

# Ubuntu Linux
sudo apt-get install <package_name>

# Mac OSX, no package manager installed by default
sudo port install <package_name>
brew install <package_name>
```

Good practices: Git (1/3)

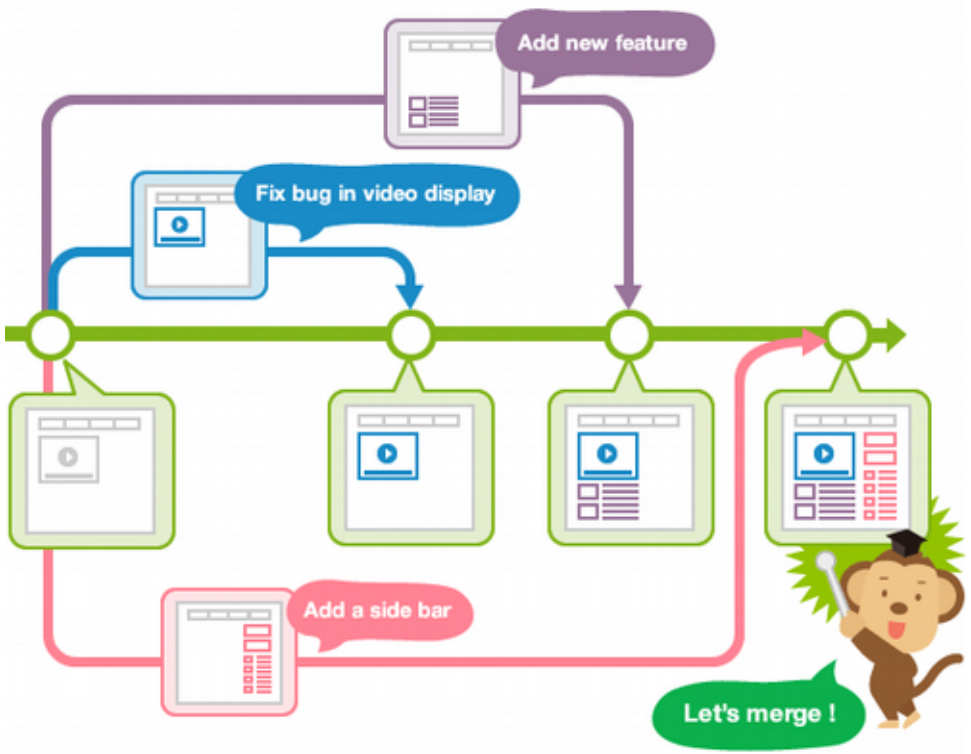
- Main purpose: **record history of your code, collaborate**
- Do not: (1) manually backup code, (2) send code around by e-mail.
- Git = a fast revision control system. Simple example:

```
# Make an project directory
make petproject
cd petproject

# Initialize git (makes and populations .git subdir)
git init

# Add some code, e.g. adder.py and record it in git history
git add adder.py
git commit -m 'First commit'

# Add another assert line to the unit test, then run:
git commit adder.py -m 'Added unit test to adder.py'

# View the history
git log

# Create account and repository on github.com, then upload the history:
git remote set-url origin git@github.com:<account_name>/<project_name>
git push origin master
```

Good practices: Git (2/3)

Some terminology

- **A commit:** one step in the history (changes in files).
- **A repository:** archive containing the history of a project.
- **A fork**: your copy of someone else's repository.
- **A branch:** a line of development, series of commits.
- **A pull request**: request to merge commits into another branch



Good practices: Git (3/3)

Official Git website: https://git-scm.com

Tutorials for beginners:

- Try here: https://try.github.io/
- https://backlogtool.com/git-tutorial/en/intro/intro1_1.html

Git hosting services (online repositories)

- http://github.com/ most popular and advanced
- http://github.ugent.be/ private github repositories
- http://gitlab.com/ less features but also less limitations

CMM Software

https://molmod.ugent.be/software

https://github.com/molmod

https://github.com/theochem

There is a lot of experience at the CMM. Ask for advice!

I'm currently actively managing:

- **Molmod**: basic molecular modeling tools
- **Yaff**: force-field molecular dynamics
- **TAMkin**: vibrational analysis and thermodynamics
- **HORTON**: experimental electronic structure code